1996

Selecting Jobs for a Heavily Loaded Shop With Lateness Penalties

Susan A. Slotnick, *Cleveland State University*
Thomas E Morton

Pergamon

0305-0548(95)00015-1

Computers Ops Res. Vol. 23, No. 2, pp. 131–140, 1996

0305-0548/96 $9.50 + 0.00

SELECTING JOBS FOR A HEAVILY LOADED SHOP WITH LATENESS PENALTIES

Susan A. Slotnick[1]† and Thomas E. Morton[2]‡

[1]W. Averell Harriman School for Management and Policy, State University of New York at Stony Brook, Stony Brook, NY 11794-3775 and [2]Graduate School of Industrial Administration, Carnegie Mellon University and Parsifal Systems, Pittsburgh, PA 15213, U.S.A.

(Received July 1992; in revised form March 1995)

Scope and Purpose—The purpose of this paper is to examine a set of trade-offs that can arise if a manufacturing facility has more potential work than it can handle easily. That is, if there are more jobs available to be processed than can be done without incurring time-related penalties such as lateness, how do we decide which ones to accept? This situation is likely to arise when the market forces firms to compete on the dimensions of service (i.e. on-time delivery) as well as price. In particular, an industry in which competition forces prices to be fairly low relative to processing costs and lateness penalties would face trade-offs between the revenue brought in by each job and the contribution of that job to lateness. In this paper, we formulate a one-machine model for this scenario, devise an optimal solution method and two heuristic approaches, and present computational results. We find that a fast, low-cost heuristic produces near-optimal results.

Abstract—One way of adjusting the workload in a manufacturing facility when available jobs exceed current capacity is to select a subset of jobs with the objective of maximizing total net profit for the firm, that is, revenue contributed by jobs processed less a penalty for lateness. This motivates us to formulate the objective function in terms of revenues minus costs, in order to reflect the trade-offs faced by the firm itself, which views costs mainly in the context of the reduction of its profits. We present a model that uses weighted lateness as a criterion for time-related penalties. Taking advantage of two special characteristics of the problem, we develop an optimal algorithm and two heuristic procedures. Computational results show that the myopic heuristic demonstrates near-optimal performance in a fraction of the processing time of the optimal benchmark.

1. INTRODUCTION

There are various strategies for dealing with an overloaded manufacturing resource—be it a bottleneck machine, work center, or the entire shop itself. Operations management textbooks list methods like adding workers or overtime hours, subcontracting to another facility, or backlogging demand to deal with a situation in which capacity limits are exceeded [1]. With regard to short-term scheduling, an overloaded resource may be considered a "bottleneck", that is, a machine that runs closer to capacity than any other machine in the shop. Since the advent of OPT [2, 3], it has become generally accepted that "babying the bottleneck" is good scheduling practice. This may be accomplished by developing heuristic prices for resources and using them to prioritize the jobs to process [4, 5], by setting due dates to distribute the workload in a way that benefits both the firms and its customers [6], or by selecting among the jobs available for processing those that would

†Susan A. Slotnick is Assistant Professor at the Averell W. Harriman School of Management and Policy at the State University of New York at Stony Brook. The current research was done while she was a graduate student at the Graduate School of Industrial Administration at Carnegie Mellon University, where she received her Ph.D. in Industrial Administration in 1994. She also holds a Ph.D. in Linguistics from Columbia University. Her research interests include heuristic and knowledge-based approaches to scheduling, and automated explanation of expert and quantitative advice-giving systems.

‡Thomas E. Morton is Professor of Industrial Administration at the Graduate School of Industrial Administration at Carnegie Mellon University and co-founder of Parsifal Systems. He received his Ph.D. from the University of Chicago. His current research interests include bottleneck dynamics, dynamic implicit heuristic pricing of shop resources, pricing systems for communication and control of man/machine planning and scheduling systems, and near-myopic approaches to stochastic systems. He has written a book on heuristic scheduling systems.

contribute the most to the profit of the firm. This last strategy is very similar to the problem of job selection that we deal with in this study.

The idea is that various customers offer a number of jobs, and the firm decides which jobs from that pool to accept for processing, based on profitability given the shop load. The trade-off that must be considered is the revenue to be gained from jobs processed versus any penalties that result from missing due dates. This motivates us to formulate the objective function in terms of revenues minus costs, in order to reflect the trade-offs faced by the firm, which views costs mainly in the context of the reduction of its profits.

The process of job selection may be illustrated by the example of a copier repair service. Let us say that there is a weekly deadline for job orders (which creates a situation of static arrivals), with the customers communicating their preferred dates for completion. From experience, the firm knows how long it will take to repair each problem, and charges according to set rates based on labor and parts. There is a discount to the customer for completing the job after the agreed-upon due-date; this discount is proportional to the lateness of the job. In addition, the firm weights each job according to the importance of the customer; this is a managerial decision that may take into account the annual volume of work provided by a certain customer (historical or projected), the criticality of certain jobs (as submitted by the customer), the potential loss of goodwill if differentiated among customers (some are more sensitive to late delivery than others), and, most importantly, the dollar value of the job. Finally, the firm attempts to decide which jobs to select for processing, by figuring out how to maximize its profit while taking account of lateness penalties. It is clear that finishing a job early will always be appreciated by the customer; this "earliness bonus" may in fact help the firm itself, for instance, by providing more time for other jobs.

This example points up the major features of the model, as well as some possible complicating factors. Because of the time-related penalties, the net profitability of any set of jobs depends on the sequencing of jobs, the resulting completion times and consequent lateness. For this reason, a scheduling model is proposed. In the simplest case, processing times, weights and per-job revenues are given; arrivals are static; weighted lateness is the criterion for time-related penalties.

1.1. Related work

Very little has been written on the question of how to select jobs for processing. As Hendry and Kingsman [7] point out, most of the work in production planning has dealt with problems that reflect make-to-stock rather than make-to-order manufacturing environments. Glaser and Hottenstein [8] consider a related question in a simulation study that examines the interactions between the firm's estimated delivery time for jobs and the market's decision to accept or refuse a bid, based on the quoted delivery promise and the past performance of the firm. In that paper, it is the market rather than the firm that makes the job acceptance decision.

Pourbabi [9] presents a model of job selection using net profit (job revenue minus operating costs) as the objective. He posits an optimal model that takes into account processing and set-up costs as well as tardiness penalties (number of tardy jobs), and allows the firm to decide to process batches of varying sizes. This model also has a two-tier price structure. If a smaller batch size than has been ordered is produced, and the order is subsequently rejected by the customer, the firm can sell the finished material on the open market at a discounted price.

Pourbabi formulates the problem as a mixed binary linear programming model which generates an optimal plan for loading an integrated manufacturing system with finite-capacity workstations, maximizing net operating profit. He notes that improved computational performance can be achieved by relaxing the integer constraint on the number of units produced, using static instead of dynamic arrivals, using only one market price, and/or ignoring setup penalties and setup times; no computational results are given.

In a sequel to this paper, Pourbabi [10] presents the job selection problem in the context of just-in-time manufacturing. A mixed binary linear programming model is proposed to find optimal lot sizes of split batches of jobs, while maximizing total net profit and ensuring that the production quantity of each job equals its demand. Software packages that may be used to solve the model are discussed; no computational results are given. Our model follows Pourbabi's lead in considering total net profit. We use lateness penalties for our operational costs, and develop heuristics to give near-optimal solutions. Our methods are based on the idea that the search space of the problem can be reduced by

partitioning the jobs under consideration into two subsets: one that contains jobs guaranteed to be included in the optimal solution, and one whose members must be considered for removal.

Woodruff [11] considers the problem of how much work to allocate to subcontractors, and the related question of how to select which jobs to subcontract. Here the objective function maximizes the value of all jobs that can be completed in-house; this includes jobs that are not allowed to be subcontracted, and "optional" jobs, that is, jobs that may be either subcontracted or processed in-house. Each job has an assigned priority value, from which is subtracted the following: holding costs, setup costs, and aggregate costs including tooling, training and special materials. The last set of costs are different for in-house processing and subcontracting. The problem is to find a set of jobs and a sequence that will maximize this expression. Since this is NP-hard, Woodruff presents a recursive algorithm that provides a lower bound on the *value* of work that must be subcontracted, and heuristics (simulated annealing and tabu search) to find approximate solutions to the job-selection and sequencing aspects of the problem.

The problem that we are solving is similar to Woodruff's, in that we employ an objective function that selects a subset of available jobs to be processed by trading off the value of jobs with certain job-related costs. However, we use lateness as our job-related cost; Woodruff employs hard deadlines, and uses due-dates to calculate holding costs rather than time-related penalties. Thus he is modeling a situation (such as a facility using JIT), in which ship dates are *not* allowed to slip, and subcontracting is used in order to help meet these deadlines. In contrast, we are looking at a problem for which the firm can deliver goods after the specified due-date at the price of losing some of its profit on that transaction.

The relationship of job-selection strategies to production planning and scheduling in an environment with set-up times and dynamic arrivals is the focus of a study by Wester *et al.* [12]. Their performance measure is utilization rate, defined here as the number of orders accepted and processed, on the condition that they can be delivered on time. Using simulation, they test three different heuristics. The "monolithic" heuristic constructs a new schedule for all of the jobs waiting for production whenever a new job arrives, and accepts the new job and new schedule if there is no resulting lateness. The "hierarchic" heuristic compares the sum of the operation times of the jobs which have already been accepted and the new job, and accepts the new job if this sum is less than a critical work content level W_c. The "myopic" heuristic uses W_c in combination with a simple priority rule to choose the next job to be processed. The problem investigated here differs from the one that we consider in that no lateness is allowed, and also because per-job revenue as it contributes to the profit of the firm is not considered.

1.2. Summary of the paper

Section 2 describes the model in detail. The simplest case is taken: one machine, static arrivals, fixed processing times, due dates and profits. The objective function maximizes total net profit (TNP), which is the sum of the revenues of all jobs minus weighted lateness penalties. Next, two special properties of this problem are exploited. First, given a set of jobs, weighted shortest processing time is known to be optimal in the minimization of weighted lateness. Second, we make use of the existence of an easily calculated "removable" set of jobs which are the only ones that need to be considered as possible candidates to be rejected when looking for the subset of the original pool that yields the highest total net profit.

In Section 3, an optimal algorithm and two heuristics are proposed, based on these two special properties. Section 4 presents the details of the computational study, and Section 5 presents our conclusions.

2. THE MODEL

2.1. Description

Consider a one-machine model with static arrivals (all jobs available at time zero), and given weights, processing times, due dates and revenues associated with each job. The objective is to maximize total net profit, which is defined as revenue of each job minus weighted lateness. Lateness (which may be negative) is completion time minus due date, all times the given weight. The objective

function is thus

$$\max \sum_{i=1}^{n} y_i[Q_i - w_i(C_i - d_i)]$$

where

i = job index; $i<j$ implies that job i precedes job j in the processing order (i.e. WSPT order), $i, j = 1, \ldots n$
n = total number of jobs in the set
y_i = 0 or 1 (job accepted or not)
Q_i = assigned revenue of job i
w_i = weight (proportional lateness discount) of job i
C_i = completion time of job i; i.e. $C_i = \Sigma_{j=1}^{i} p_j$, where p_j is the processing time of job j
d_i = due date of job i.

2.2. Special properties

It is a known result [13] that weighted shortest processing time (WSPT) order minimizes weighted lateness. That is, if we ignore the selection issue, placing jobs in increasing order of the ratio of processing time to weight will result in an optimal schedule. The algorithm and heuristics presented here take advantage of this result.

A special property of this particular problem is that the initial set of candidate jobs may be partitioned into two subsets: one containing jobs that are easily seen to be part of the optimal solution, and the other containing jobs that may or may not be accepted into the optimal set. All procedures in this paper exploit this property in order to reduce computation by limiting the number of jobs that we consider removing, and so limiting the number of sets that we need to consider when seeking the optimal solution.

The second subset, which we will call the removable set, is found in the following manner. First, the initial set $\mathscr{S}$ is sequenced in WSPT order. The total net profit of $\mathscr{S}$, TNP($\mathscr{S}$), is calculated. Next, a single job i is removed from $\mathscr{S}$ to create n subsets $\mathscr{S}_i$, each with $n-1$ elements. Total net profit for each, TNP($\mathscr{S}_i$), is calculated. If TNP($\mathscr{S}$)<TNP($\mathscr{S}_i$), then job i is a member of the removable set; otherwise it is guaranteed to be a member of the optimal set.

Lemma 1. Let $\mathscr{T} \subseteq \mathscr{S}$, and $\mathscr{T}_i \equiv \{\mathscr{T} - \{i\}\}$.

Define $\Delta_i(\mathscr{T}) = \text{TNP}(\mathscr{T}_i) - \text{TNP}(\mathscr{T})$, *and* $\Delta_{im}(\mathscr{T}) = \Delta_i(\mathscr{T}_m) = \text{TNP}(\mathscr{T}_{im}) - \text{TNP}(\mathscr{T}_m)$ *(where* $\mathscr{T}_{im} = \mathscr{T}_m - \{i\}$*). Then* $\Delta_{im}(\mathscr{T}) \leqslant \Delta_i(\mathscr{T})$, *implying that if* $\mathscr{U} \subseteq \mathscr{S}$ *then* $\Delta_i(U) \leqslant \Delta_i(\mathscr{S})$.

Intuitively, the lemma states that removing job i from a larger subset will always result in a bigger improvement in total net profit than removing it from a smaller subset.

Proof of lemma

$$\Delta_i(\mathscr{T}) = -Q_i + w_i \sum_{k=1}^{i} p_k + p_i \sum_{k=i+1}^{n} w_k - w_i d_i. \tag{1}$$

For $i>m$:

$$\Delta_{im}(\mathscr{T}) = -Q_i + w_i \sum_{k=1,k\neq m}^{i} p_k + p_i \sum_{k=i+1}^{n} w_k - w_i d_i. \tag{2}$$

For $i<m$:

$$\Delta_{im}(\mathscr{T}) = -Q_i + w_i \sum_{k=1}^{i} p_k + p_i \sum_{k=i+1,k\neq m}^{n} w_k - w_i d_i \tag{3}$$

$$\Delta_i(\mathscr{T}) - \Delta_{im}(\mathscr{T}) = \begin{cases} w_i p_m & \text{if } i \text{ is after } m \text{ in WSPT order} \\ p_i w_m & \text{if } i \text{ is before } m \text{ in WSPT order.} \end{cases} \tag{4}$$

Since $w_j \geqslant 0$ and $p_j \geqslant 0$ for all j, $\Delta_{im}(\mathscr{T}) \leqslant \Delta_i(T)$.

This can easily be extended, one job at a time, to the case where the two subsets differ by more than one job. ■

Theorem 1. *Given an initial set of jobs S as described above, with subset* $\mathscr{S}_i = \{\mathscr{S} - \{i\}\}$, *if* $\text{TNP}(\mathscr{S}) \geqslant \text{TNP}(\mathscr{S}_i)$, *then i is a member of the optimal set with regard to profit maximization using weighted lateness as time-related penalty*

Proof of Theorem. We will assume that there exists a job i, not a member of the optimal set, such that $\text{TNP}(\mathscr{S}) \geqslant \text{TNP}(\mathscr{S}_i)$, and show that a contradiction results. Let $\mathscr{V}$ be the optimal set, and $\mathscr{N} = \mathscr{S} - \mathscr{V}$ be the members of $\mathscr{S}$ not in the optimal set. Now suppose there exists a job $i \in \mathscr{N}$, where $\mathscr{N} = \{i, n_1, n_2, \ldots, n_k\}$. By assumption, $\text{TNP}(\mathscr{S}) \geqslant \text{TNP}(\mathscr{S}_i)$, which means that $\Delta_i(\mathscr{S}) \leqslant 0$. From the lemma, $0 \geqslant \Delta_i(\mathscr{S}) \geqslant \Delta_{in_1}(\mathscr{S})$, so $\text{TNP}(\mathscr{S}_i) \geqslant \text{TNP}(\mathscr{S}_{in_1})$. On the same lines, $\text{TNP}(\mathscr{S}_{n_1}) \geqslant \text{TNP}(\mathscr{S}_{in_1}) \geqslant \ldots \text{TNP}(\mathscr{S}_{in_1 \ldots n_k}) = \text{TNP}(\mathscr{S} - \mathscr{N}) = \text{TNP}(\mathscr{V})$. This is a contradiction since $\text{TNP}(\mathscr{V})$ is the total net profit of the optimal set which is by definition the largest TNP of any subset. A formal proof by induction is straightforward; it is omitted here. ■

3. SOLUTION PROCEDURES

The algorithm and heuristics presented here take the above two properties into account. In each case, the full initial set of jobs ($\mathscr{S}$) is first arranged in WSPT order. Next, the removable set of jobs ($\mathscr{R}$) is found. Finally, the optimal solution is sought using a reduced combinatorial "top-down" method: beginning with the initial set, subsets are created by deleting "removable" jobs; the subset found with the highest total net profit is the solution. The procedures differ principally in the way in which evaluation of subsets is limited.

3.1. Branch-and-bound algorithm with linear relaxation

In order to find an optimal solution to this problem, we use a branch-and-bound algorithm, in which the bounding is provided by a linear relaxation of the current problem at each node. Profit values of nodes in the decision tree are non-monotonic, since the value of each node (total net profit) is calculated by subtracting any lateness penalties from the potential profit value of the set of jobs being considered. If the problem is formulated as an integer program, and the integer requirement is then relaxed, a linear programming solution can be found for each node. Since the child nodes all contain fewer jobs than their parents, the values generated by the linear programs are monotonic, i.e. they can only decrease toward leaf nodes. These values serve as upper bounds and enable pruning of any branch whose linear-programming solution has a profit value that is lower than that of the best solution found so far.

We reformulate the problem as a mixed integer linear program in the following manner. We write the original objective function as

$$\max \sum_{i=1}^{n} y_i[Q_i - w_i(C_i - d_i)].$$

The expression to be summed multiplies out to the following: $y_iQ_i - y_iw_iC_i + y_iw_id_i$. Since $C_i = \Sigma_{j=1}^{i} p_jy_j$, the second term of this expression is quadratic. Let $x_i = y_iC_i$, that is, x_i is the completion time if the job is accepted, and zero otherwise. Now the expression above may be rewritten as $y_iQ_i - w_ix_i + y_iw_id_i$. The decision variables become x_i and y_i, and the second term is now linear in these variables. The reformulated problem becomes

$$\max \sum_{i=1}^{n} Q_iy_i - w_ix_i + d_iw_iy_i$$

subject to

1. $x_i \leqslant M_iy_i$ (where $M_i = \Sigma_{j=1}^{i} p_jy_j$, the maximum possible completion time of job i).
2. $x_i + M_i(1 - y_i) \geqslant \Sigma_{j=1}^{i} p_jy_j$.
3. $x_i \geqslant p_iy_i$.
4. $y_i = 0$ or $y_i = 1$.
5. $x_i \geqslant 0$, $y_i \geqslant 0$.

The first constraint ensures that, if job i is accepted, its completion time will not exceed its maximum possible value; if the job is not accepted, this constraint ensures that $x_i = 0$. The second constraint guarantees that the completion time of job i will be adjusted for any job before it that is not accepted. For each job i, either constraint (1) or constraint (2) will be binding, depending on whether the job has been accepted or not. Constraint (3) guarantees that $y_i = 0$ if $x_i = 0$, and also that if the job is accepted, its completion time will be at least as large as its processing time. In addition to enforcing feasibility, these three constraints keep the bounds tight when the integer constraint on y_i is relaxed. The second formulation is equivalent to the first,† and generates the same optimal results. To use this formulation, the program first sets $y_i = 1$ for each job that is known to be in the optimal set (i.e. not in the removable set). Then the integer variable is relaxed.

For an illustration of how this bounding method works, see Fig. 1. The top node in the figure represents the scheduling problem with the full set of available jobs; it has a total net profit (TNP) of 18, and an upper bound (UB, as calculated by the linear relaxation) of 32. This means that if all the jobs were put in WSPT order, the resulting profit would be 18. If the integer constraint were relaxed (that is, jobs could be processed in arbitrarily small chunks), the profit would be 32. The next level of nodes represents three new problems, each generated by removing a different job from the original problem; each subsequent level is generated in the same manner. Now assume that we have generated a trial solution of TNP = 29, which is the best solution that we have so far. Any node with an upper bound less than 29 can be pruned, since we know that its TNP, and that of its children, will never be as good as the trial solution. As a result, we can prune nodes 5, 6 and 10.

Because calculating the linear relaxation is computationally expensive, we first prune the search tree by using the following dominance property. By applying Theorem 1 to the subset of jobs at each node, we see that, if the removal of a job results in a subset with profit lower than the profit of its parent node, then the TNP of all child nodes can only be lower. As a result, that branch is fathomed and we can remove it from further consideration. However, it is not true that this job is guaranteed to be part of the optimal set; so a stronger version of this dominance property, which also removes such a job from further consideration as part of the removable set *across the entire tree*, generates sequences that are not guaranteed to be optimal. Looking again at Fig. 1, we can prune nodes 5 and 10 using this dominance property.

The branch-and-bound algorithm works as follows:

1. Find a trial solution using a good heuristic (we use the myopic heuristic described below, which is very quick and usually gives the optimal solution or very close to it).

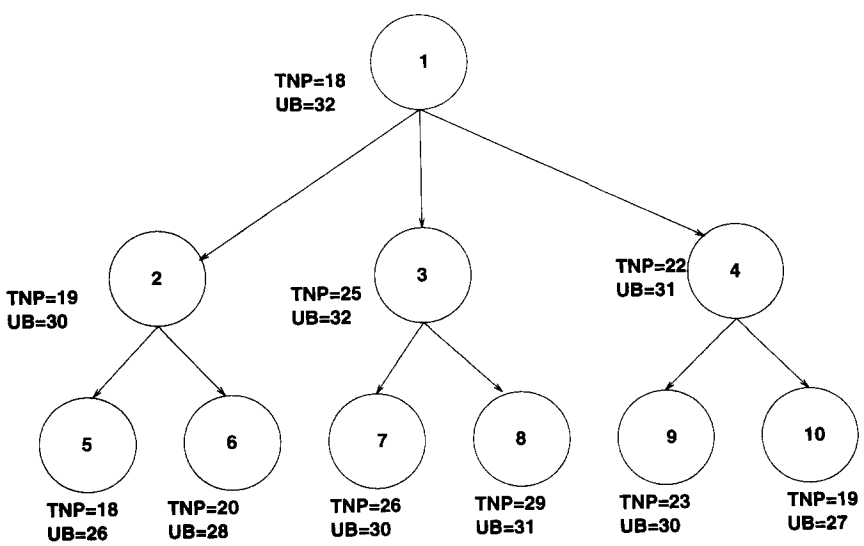

Fig. 1. Example of bounding by linear relaxation and dominance property.

†A proof will be supplied by the authors upon request.

2. Put jobs in WSPT order. Find the removable set.
3. Work down through the tree, generating nodes by deleting successive "removable" jobs.
4. Compare the net profit for the current node with the net profit of its parent node. If its net profit is lower than that of its parent, delete it. It has been fathomed.
5. If the current node passes the above dominance test, compute its linear-programming solution. If the value of its net profit is lower than the trial solution, prune this branch.
6. Replace the trial solution with any better solution that is found in the course of processing.
7. The procedure terminates when all branches have been fathomed.

The potential size of the generated tree is dependent upon how many jobs are removable. If calculation of duplicate subsets can be eliminated, computation is thus $O(2^r)$, where r is the cardinality of the removable set. In order to avoid duplicate calculations, the branch-and-bound algorithm incorporates an indexing routine that keeps track of which subsets have already been evaluated.†

3.2. Beam search heuristic

Because in our studies the branch-and-bound algorithm becomes unwieldy rather quickly, we turned to a beam search formulation in order to provide a good benchmark for comparison with the myopic heuristic for larger problems. Beam search is an approach that limits the generation of the search tree to some predetermined number of "most promising" branches at each level. It was developed by researchers in the field of artificial intelligence, and has been previously applied to scheduling problems by Ow and Morton [15] and Chang, Matsuo and Sullivan [16].

The beam search heuristic for this problem works as follows:

1. Put jobs in WSPT order. Find the removable set.
2. Work down through the tree, generating nodes by deleting successive "removable" jobs. For each node generated, evaluate its "promise" by calculating the solution of the myopic heuristic on this subset of jobs. For each level of the tree, prune all but the most promising b nodes.
3. Keep track of the best solution that is generated in the course of processing.
4. The procedure terminates when all branches have been fathomed.

Computation for beam search is $O(br^4)$, where r is the size of the removable set and b is the beamwidth (see Ow and Morton [15] for a discussion of the computational complexity of beam search). Since we were using beam search as a substitute benchmark for the optimal algorithm, we tested it against the branch-and-bound algorithm on a set of one hundred 17-job problems, with the beam width set at 17. Beam search found the optimal solution in every case, at considerable savings in processing time.

3.3. Myopic heuristic

The myopic heuristic pursues only the path that looks currently most promising. It generates a much smaller tree, since it simply removes the last profitable job at each node until only a single job can be removed to improve the total net profit. As soon as the total net profit of all subsets on one level does not improve on the best of the previous level, the procedure terminates. Computation is $O(r^2)$; in fact, the largest number of nodes generated is $[r + (r-1) + (r-2) + \cdots + 1]$.

The details of this heuristic are as follows:‡

1. Put jobs in WSPT order.
2. Find the job with the highest $\pi_i = p_i \Sigma_{j=i+1}^{n} w_j + w_i(C_i - d_i) - Q_i$.
3. If $\pi_i \leqslant 0$, terminate.
4. Otherwise remove job i.
5. Return to step 2.

†Baker [14] presents a similar indexing convention in his FORTRAN code for a dynamic programming algorithm.

‡We would like to thank an anonymous referee for suggestions which led to this version of the myopic heuristic. The idea of the removable set is implicit in this version of the heuristic.

The myopic heuristic works exceptionally well for the type of problem we consider here, since the combination of any one job's contribution to average lateness and net profit is a good rough indicator of whether or not it should be chosen for processing. Intuitively, jobs with relatively short processing times and large contributions to total profit should be preferred to jobs that take a longer time to complete and do not have as much payoff. In typical terms, in which processing times, weights and due dates are relatively uniform, the "bad" jobs can be determined at a fairly early stage of the search tree.

4. COMPUTATIONAL STUDY

The myopic heuristic was tested on four hundred problems, one hundred each with 12, 17, 22 and 27 jobs. For 12- and 17-job problems, the benchmark was the optimal branch-and-bound algorithm. For the larger problems, solutions generated by the beam search heuristic provided the benchmark.

Test programs were coded in FORTRAN77, and all tests were run on a DECstation 5000 under the UNIX operating system. Weight, processing time, due date and revenue were randomly generated. The first three were real numbers drawn uniformly from [0,10], and job revenue was a randomly generated real number from a lognormal distribution with an underlying normal distribution with mean 0 and standard deviation 1.† This reflects a situation in which job characteristics are fairly similar, but where the potential revenue of any job may vary widely (see Morton *et al.* [4]). For the beam search procedure, the beamwidth was equal to the number of jobs in the set.

Tables 1 and 2 show aggregate results for these tests. Tests are displayed in ascending order of the size of the problems. As shown in the columns comparing CPU times, the myopic heuristic was always much faster than the benchmark methods. It was also extremely accurate, finding the benchmark solution in all but 65 of the 400 examples generated. Over all four hundred problems, the myopic solutions diverged on average 1.30% from the benchmark solution. The worst single case diverged 51% from the benchmark solution.

From Tables 1 and 2, we see that, as we might expect, larger problems take both procedures longer to process. Within each group of problems (consisting of 12, 17, 22 and 27 jobs, respectively), those with larger removable sets results in longer processing times; within problems with the same size of removable set, those with smaller solution sets result in longer processing times. One way of determining "difficulty" of these problems, then, is to look first at the size of the removable set (bigger is harder), and then to look at the size of the solution set (smaller is harder). There is also a rough correlation between these factors and the accuracy of the myopic heuristic.

In order to generate problems that were as difficult as possible, with respect to the sizes of the removable set and solution set as described above, we used numbers for job revenue that are relatively small. This reflects a very tight profit margin for the firm; it is representative of an industry in which competition forces prices to be fairly low relative to processing costs and lateness penalties. To test how our methods would work in a situation where per-job revenue was higher relative to processing costs, we ran a study of fifty 20-job problems in which the revenue was generated to be roughly five times the weight. These problems were much easier, in that the size of the removable sets were much smaller than the size of the solution sets (the removable sets ranged from 1 to 9 jobs, while the solution sets contained from 14 to 19 jobs). As expected, we found that the myopic heuristic generated the optimal solution in every case, with considerable savings in computation time. The average time for the branch-and-bound algorithm was 1,230.411 s of CPU time, while the average for the myopic heuristic was 0.016 s; the slowest time for myopic heuristic was 0.031 s, and for the branch-and-bound algorithm, 37,273.98 s; the fastest time for the myopic heuristic was 0.008 s, and for the branch-and-bound algorithm, 0.437 s.

†Note that these distributions guarantee that all numbers generated are positive. We used standard routines from the IMSL libraries: RNUN for the uniform distribution, and RNLNL for the lognormal distribution. For the latter, the probability density function is defined as:

$$f(x) = \frac{1}{\sigma x\sqrt{2\pi}} \exp\left[-\frac{1}{2\sigma^2}(\ln x - \mu)^2\right] \quad \text{for } x > 0$$

For more details, see [17].

Table 1. Aggregate results for 400 experiments (12- and 17-job problems)

No. jobs	No. jobs in removable set	No. jobs in benchmark solution	No. problems	CPU time (s)		Average % deviation
				Bench.	Myopic	
12	12	1	2	9.11	0.01	0.00
12	12	2	24	7.92	0.01	2.63
12	12	3	17	7.04	0.01	0.55
12	12	4	4	6.06	0.01	1.59
12	12	5	1	4.82	0.01	0.00
12	11	2	9	4.23	0.01	0.23
12	11	3	11	4.05	0.01	0.00
12	11	4	13	4.24	0.01	0.03
12	11	5	3	4.90	0.01	1.49
12	11	6	2	7.45	0.01	0.00
12	10	4	8	1.80	0.01	0.00
12	10	5	3	2.13	0.01	0.00
12	9	6	3	1.28	0.01	0.00
17	17	1	1	689.33	0.04	0.00
17	17	2	7	700.68	0.04	12.38
17	17	3	23	575.61	0.03	0.91
17	17	4	16	563.19	0.03	0.22
17	17	5	13	452.16	0.03	0.15
17	16	2	2	355.83	0.04	0.63
17	16	3	3	318.69	0.03	12.20
17	16	4	15	279.76	0.04	2.13
17	16	5	6	251.97	0.03	0.00
17	16	6	4	204.77	0.03	0.00
17	15	3	1	149.17	0.04	5.26
17	15	4	2	151.05	0.03	2.24
17	15	5	2	144.56	0.03	0.00
17	15	6	1	76.66	0.02	0.00
17	14	5	1	58.63	0.02	6.09
17	14	6	1	70.55	0.02	0.00
17	14	7	1	81.41	0.02	0.00
17	13	6	1	25.99	0.02	0.00

Table 2. Aggregate results for 400 experiments (22- and 27-job problems)

No. jobs	No. jobs in removable set	No. jobs in benchmark solution	No. problems	CPU time (s)		Average % deviation
				Bench.	Myopic	
22	22	2	4	127.90	0.08	4.87
22	22	3	14	129.60	0.08	5.85
22	22	4	19	125.82	0.08	2.53
22	22	5	14	123.47	0.07	0.61
22	22	6	4	115.58	0.07	0.98
22	22	7	1	137.24	0.09	0.00
22	21	3	1	134.04	0.09	0.00
22	21	4	9	110.13	0.07	1.10
22	21	5	9	107.03	0.07	1.09
22	21	6	11	100.72	0.07	0.06
22	20	4	2	101.51	0.07	0.28
22	20	5	3	96.99	0.07	0.00
22	20	6	4	88.60	0.06	0.16
22	20	7	3	92.02	0.07	0.00
22	19	6	1	89.74	0.07	0.00
22	17	9	1	46.47	0.05	0.00
27	27	2	1	549.67	0.17	0.56
27	27	3	13	462.96	0.14	2.92
27	27	4	16	440.16	0.14	0.00
27	27	5	17	441.05	0.14	0.08
27	27	6	12	417.53	0.13	0.20
27	27	7	3	398.53	0.13	0.00
27	27	8	3	386.01	0.12	0.21
27	26	4	7	402.84	0.13	0.75
27	26	5	7	383.62	0.13	0.82
27	26	6	8	368.17	0.12	0.00
27	26	7	7	357.38	0.12	0.10
27	25	5	1	329.19	0.12	0.00
27	25	6	1	335.27	0.12	0.00
27	25	7	1	313.41	0.11	0.00
27	25	8	1	310.13	0.11	0.00
27	24	6	1	285.57	0.11	0.00
27	24	8	1	313.28	0.13	0.00

5. CONCLUSIONS

This model of job selection in the context of heavy shop load presents a problem the structure of which is ideal for the type of myopic heuristic that we tested here. Because we maximize total net profit (total profit minus lateness costs) rather than minimizing lateness, optimal procedures like branch-and-bound are computationally very expensive (due to the dimensionality of the search space as well as the type of bounding method, such as linear programming, that must be used). Although it is not optimal to remove a job from consideration once it has been shown that its removal will improve net profit at some intermediate stage, this usually works for typical problems in which the jobs have similar processing characteristics. Both the beam search and myopic heuristic exploit this property, saving only the best results as they work their way through the search tree. These heuristics run in a fraction of the time taken by the optimal algorithm, with results that are optimal or very close to it most of the time. Possible extensions of this study include investigation of how the concept of the removable set applies to other classes of objective functions, and how it might be adapted to problems with different characteristics such as dynamic arrivals, set-up times or precedence constraints.

Acknowledgements—We wish to thank Kenneth R. Baker, Sunder Kekre, Prasad Ramnath and an anonymous referee for their helpful suggestions.

REFERENCES

1. J. R. Meredith, *The Management of Operations: A Conceptual Emphasis*. Wiley, New York (1992).
2. R. Lundrigan, What is this thing called OPT? *Product. Invent. Mgmt* **27,** 2–11 (1986).
3. M. P. Meleton, OPT-fantasy or breakthrough? *Product. Invent. Mgmt* **27,** 13–21 (1986).
4. T. E. Morton, S. R. Lawrence, S. Rajogopalan and S. Kekre, SCHED-STAR: A price-based shop scheduling module. *J. Manufact. Opns Mgmt* **1,** 131–181 (1988).
5. T. E. Morton and D. W. Pentico, *Heuristic Scheduling Systems. With Applications to Production Engineering and Project Management*. Wiley, New York (1993).
6. T. C. E. Cheng and M. C. Gupta, Survey of scheduling research involving due date determination decisions. *Eur. J. Opl Res.* **38,** 156–166 (1989).
7. L. C. Hendry and B. G. Kingman, Production planning systems and their applicability to make-to-order companies. *Eur. J. Opl Res.* **40,** 1–15 (1989).
8. R. Glaser and M. Hottenstein, Simulation study of a closed-loop job shop. *J. Opns Mgmt* **2,** 155–166 (1982).
9. B. Pourbabi, A short term production planning and scheduling model. *Engng Costs Product. Econ.* **18,** 159–167 (1989).
10. B. Pourbabi, Optimal selection of orders in a just-in-time manufacturing environment: a loading model for a computer integrated manufacturing system. *Int. J. Comput. Integrated Manufact.* **5,** 38–44 (1992).
11. D. L. Woodruff, Subcontracting when there are setups, deadline and tooling costs. In *Proc. Intelligent Scheduling Systems Symp.* (Edited by Scherer W. T. and Brown D. E.), pp. 337–353 (1992).
12. F. A. W. Wester, J. Wijngaard and W. H. M. Zijm, Order acceptance strategies in a production-to-order environment with setup times and due-dates. *Int. J. Product. Res.* **30,** 1313–1326 (1992).
13. W. E. Smith, Various optimizers for single-stage production. *Naval Res. Logist. Q.* **3,** 59–66 (1956).
14. K. R. Baker, *Introduction to Sequencing and Scheduling*. Wiley, New York (1974).
15. P. S. Ow and T. E. Morton, Filtered beam search in scheduling. *Int. J. Product. Res.* **26,** 35–62 (1988).
16. Y-H. Chang, H. Matsuo and R. S. Sullivan, A bottleneck-based beam search for job scheduling in a flexible manufacturing system. *Int. J. Product. Res.* **27,** 1949–1961 (1989).
17. IMSL, *User's Manual: FORTRAN Subroutines for Statistical Analysis*, Chap. 18, Random number generation. IMSL (1991).