Scheduling Jobs with Simple Precedence Constraints on Parallel Machines

Debra J. Hoitomt, Peter B. Luh, Eric Max, and Krishna R. Pattipati

ABSTRACT: This article presents a methodology for scheduling jobs on identical, parallel machines. Each job is comprised of a small number of operations that must be processed in a specified order. The objective is to minimize the total weighted quadratic tardiness of the schedule, subject to capacity and precedence constraints. The procedure presented here is an efficient near-optimal method based on the Lagrangian relaxation technique and the list-scheduling concept. In addition, the resulting job-interaction information can be used to provide quick answers to "what-if" questions and to reconfigure the schedule to incorporate new jobs and other dynamic changes. This scheduling methodology has been implemented for a work center at Pratt & Whitney as part of its knowledge-based scheduling system. Typical sizes of problems involve 35 to 40 machines and 100 to 200 jobs, each with 3 to 5 operations.

Introduction

The Scheduling Problem

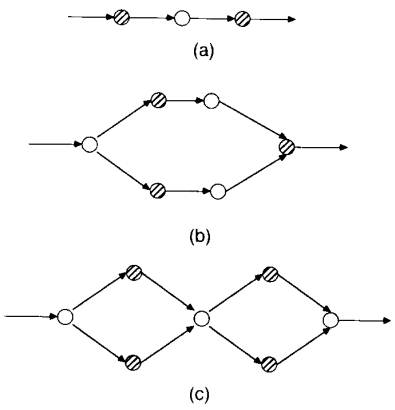
Many productivity problems in the United States are associated with scheduling operations. Finding methodologies that consistently generate "good" schedules, however, has frustrated researchers and practitioners alike. The difficulty is that most scheduling problems belong to the class of NP-hard combinatorial problems, for which the development of efficient optimum-producing polynomial algorithms is unlikely. In view of the increasing complexity of production operations, scheduling problems, even those faced by small- to medium-sized companies, are beyond the reach of most existing mathematical techniques. Therefore, practical schedules are commonly generated by simple heuristic algorithms with questionable performance or expensive computer simulation with questionable validity. The interaction of jobs, as they compete for limited resources, is not visible; dynamic changes in the system (for example, machine breakdown, jobs requiring more time than anticipated, etc.) cannot be easily accommodated; and job completion dates cannot be accurately predicted or controlled. Consequently, jobs often have long lead times, and shops are loaded with excessive work-in-process inventories. Therefore, the resolution of these difficulties can translate into considerable direct and indirect cost savings.

In this article, we consider the nonpreemptive scheduling of jobs with due dates on identical, parallel machines. Each job is made up of a small number of operations that must be undertaken in a particular order. The precedence constraints or process plans are restricted to those that can be represented by a simple fork/join type of graph as the one shown in the illustration. In this graph, each operation is represented by a node, and the "forking" represents the case where several operations may be performed simultaneously. The nesting of forks and joins within a fork/join will not be considered here. There may be a "time-out" between successive operations for inspection, heat treatment, paperwork, or other processing that does not require the use of the machines under consideration. The time requirements for processing and time-outs are assumed to be known. Jobs also have different due dates and different levels of importance. The objective is to minimize the total weighted quadratic tardiness of the schedule.

This problem is representative of a bottleneck work center in a job shop, where time-outs represent visits to other nonbottleneck work centers. Recent manufacturing philosophy suggests that bottlenecks implicitly control the flow of jobs in a shop [1]. Thus, an effective and efficient solution to this problem has far-reaching implications.

Research on Relevant Scheduling Problems

A parallel machine-scheduling problem with precedence constraints is studied in [2] and [3]. The objective function considered is the maximum completion time (makespan). This problem is shown to be NP-hard, along with many other scheduling problems with precedence constraints. Therefore, at least partial enumeration of possible solutions is required to determine the best or optimal solution, and solution methodologies are usually costly and inefficient for large problems.



Typical precedence structures of the simple fork/join type.

An approach that has proven useful in large industrial settings is the development of heuristics. Heuristics are generally more efficient than analytical approaches (e.g., branch and bound) [4]. Many heuristics are based upon the list-scheduling concept. Jobs are ordered by some criterion and scheduled according to this list on the first available machine. By using makespan as the objective function without precedence constraints, it has been shown that the worst-case performance ratio (i.e., the ratio of a heuristic cost to the optimal cost) for any generic list-scheduling algorithm is $(2 - 1/m)$, where m is the number of machines [5]. When jobs are independent and the longest processing time (LPT) ordering (i.e., jobs arranged in decreasing order of processing times) is used, this bound reduces to $[4/3 - 1/(3m)]$ [6]. Tardiness criteria generally complicate the situation [3], and the presence of precedence constraints implies even worse bounds [5].

Debra J. Hoitomt and Eric Max are with Pratt & Whitney, East Hartford, CT 06108; Peter B. Luh and Krishna R. Pattipati are with the Department of Electrical and Systems Engineering, University of Connecticut, Storrs, CT 06269-3157.

0272-1708/90/0200-0034 $01.00 © 1990 IEEE

In addition, the scheduler generally has no idea about how any particular schedule scores, whether close to the worst case or near to optimal.

Statistical optimization (or simulated annealing) procedures present a new and promising approach to the scheduling problem [7], [8]. In these methods, a nominal schedule or set of schedules is varied in a small and generally random way. A probability, which is determined by the relative change in the schedule cost, is assigned to the result. This probability is then used to determine which schedule or set of schedules becomes the nominal for the next iteration. As in most other optimization methods, there is some degree of enumeration, and the methods are generally slow to converge to the optimum. Similar to most heuristic methods, it is difficult to determine when or if an optimum has been attained. If the algorithm is halted before an optimal schedule is generated, there is no way to measure the quality of the resulting schedule.

It is generally concluded that a gap exists between scheduling theory and practice [4]. Practical methods react to dynamic changes without the ability to reliably produce a good schedule, and theoretical methods produce good schedules without the ability to react to dynamic changes. In a recent effort, we considered the scheduling of single-operation jobs on parallel machines [9], [10]. A demonstrably efficient and near-optimal algorithm based on the Lagrangian relaxation technique was presented. The dual solution provides an execution order for the list-scheduling method. The dual cost is also a lower bound on the performance, indicating near-optimal performance for most of the cases tested (within 1 percent of lower bound). Furthermore, the optimal Lagrange multipliers provide valuable job-interaction information to answer "what-if" questions and to reconfigure the schedule in order to incorporate new jobs and other dynamic changes in the system (e.g., machine breakdowns, jobs requiring more time than anticipated).

By extending our results [9], [10], the scheduling of jobs with precedence constraints on parallel machines is considered in [11]. The problem is solved by imposing an early start time and a due date for each operation in the precedence structure so that operations cannot overlap. The research reported in this article is also based on our previous work [9], [10]. However, it presents an alternative approach, in that operations are restricted only by the precedence structure without artificial early start times and due dates. In addition, the optimal Lagrange multipliers are used to efficiently reconfigure the schedule to accommodate dynamic changes and also to schedule new jobs.

Problem Formulation

An integer programming formulation is a common way to represent a scheduling problem. The discrete-time integer programming formulation developed here follows the model of [9], [10] and has been influenced by the work of [12] in some variable definitions, [13] in some constraint statements, and generally by [14]–[16].

The quantity to be minimized J is the sum of a weighting w_i times the square of the tardiness T_i (time past the due date) for each job, where the subscript i refers to the ith job.

$$J \equiv \sum_i w_i T_i^2 \tag{1}$$

Our previous papers [9], [10] used the sum of weighted tardiness T_i rather than weighted quadratic tardiness used here. For the weighted tardiness function, the incremental penalty of a job does not change as the tardiness increases. Thus, for example, for two jobs with weight 1, both jobs one day late (objective function $J = 2$) is equivalent to one job two days late ($J = 2$). The weighted quadratic tardiness function resolves this ambiguity. Here, two jobs one day late has lower cost ($J = 2$) than the situation of one job being two days late ($J = 4$). This tardiness objective function accounts for the values of jobs, the importance of meeting due dates, and the fact that a job becomes more critical with each time unit after passing its due date. Choosing the weights is a judgmental decision. Note that a job with weight 1 and x days late has the same penalty as a job with weight x^2 and a one-day lateness.

Each job is decomposed into a series of operations with b_{ij}, defined as the beginning time of the jth operation of the ith job (referred to as operation $[i, j]$). The beginning times b_{ij} belong to an eligible set B_{ij}, with smallest and largest allowable beginning times determined by the early start time of the job, processing time requirements of operations of job i, and the time horizon. The scheduling problem is to minimize the weighted quadratic tardiness J subject to the following three constraints: (1) precedence constraints, (2) capacity constraints, and (3) processing time requirements.

The precedence constraints are that the operations must be performed in the order dictated by the process plan graph. The equation is that the completion time c_{ij} plus the slack time s_{ij} after operation (i, j) must be less than the beginning times of all operations l, which immediately follow operation (i, j).

$$c_{ij} + s_{ij} + 1 \leq b_{il} \tag{2}$$

The equation specifying the capacity constraints states that the total number of operations being processed (active) at time k must be less than or equal to the number of machines M_k available at time k. The nonnegative integer δ_{ik} represents the number of operations of job i active at time k.

$$\sum_i \delta_{ik} \leq M_k \tag{3}$$

The equation for the process time requirements t_{ij} states that the elapsed difference between the beginning time b_{ij} and the completion time c_{ij} should be t_{ij}.

$$c_{ij} - b_{ij} + 1 = t_{ij} \tag{4}$$

In the above formulation, the time horizon, the number of jobs, the weights of jobs, precedence structures, processing time requirements, slack times, due dates, and machine availability are assumed to be given. Decision variables are the beginning times of operations b_{ij}. Once beginning times are selected, completion times, number of active operations at a given time, and tardinesses can be easily derived. For example, for the job depicted in part (a) of the illustration with $b_{i1} = 2$, $t_{i1} = 3$, $b_{i2} = 8$, $t_{i2} = 2$, and $D_i = 8$, we have $\delta_{i2} = \delta_{i3} = \delta_{i4} = 1$, $\delta_{i8} = \delta_{i9} = 1$; $C_i = c_{i2} = 9$, and $T_i = C_i - D_i = 1$. Note that precedence constraints and processing time requirements relate to individual jobs, and the objective function is also a job-wise additive. Only capacity constraints couple across jobs and make the problem intractable.

Solution Methodology

The Lagrangian Relaxation Approach to the Scheduling Problem

Lagrangian relaxation has often been employed to relax coupling constraints to obtain a set of decomposed subproblems from an original problem [17]. This is clearly desirable if subproblems are easier to solve than the original one. The approach was pioneered by [14], while [15] recognized the potential specifically for scheduling problems. The theoretical aspects for integer programming problems and the corresponding solution methodology were further developed in [18]. Recently, the relaxation procedure has been exploited to obtain solutions for related problems [16], [19], [20].

To decompose the problem according to jobs, we relax the coupling capacity con-

straint (3) by using the nonnegative Lagrange multiplier π_k and form the following Lagrangian R.

$$R \equiv J + \sum_k \pi_k \left(\sum_i \delta_{ik} - M_k \right)$$
$$= -\sum_k \pi_k M_k + \sum_i L_i \qquad (5)$$

where

$$L_i \equiv \left\{ w_i T_i^2 + \sum_k \pi_k \delta_{ik} \right\} \qquad (6)$$

This leads to a decomposed subproblem for each job i (given $\{\pi_k\}$), where L_i is to be minimized with respect to the beginning times $\{b_{ij}\}$ subject to precedence constraints (2) and processing time requirements (4) of job i. Denoting the minimum of (6) for the particular values of $\{\pi_k\}$ by $L_i^*(\pi)$, the dual problem is then to maximize the function L with respect to π.

$$L(\pi) \equiv -\sum_k \pi_k M_k + \sum_i L_i^*(\pi) \qquad (7)$$

The above derivation presents a decomposition framework for solving the scheduling problem. As in [9], [10], there are several steps to obtain a near-optimal solution: solving subproblems, solving the dual problem, and constructing a feasible solution. The precedence constraints (2), however, necessitate different solution methodologies for each step. We shall briefly discuss each of them.

Scheduling Individual Jobs

In scheduling job i with a given set of multipliers $\{\pi_k\}$, we note that capacity constraints have been relaxed, while precedence constraints and processing time requirements for the job still hold. If operation (i, j) begins on a machine at time b_{ij}, then it ends at time $c_{ij} = b_{ij} + t_{ij} - 1$. Define L_{ij} as the cost of performing operation (i, j) over the interval $[b_{ij}, c_{ij}]$.

$$L_{ij} \equiv \sum_{k=b_{ij}}^{c_{ij}} \pi_k \qquad (8)$$

Then (6) can be rewritten as the sum of the weighted quadratic tardiness and the cost of processing the operations of job i.

$$L_i = w_i T_i^2 + \sum_{j=1}^{N_i} L_{ij} \qquad (9)$$

This problem may be solved by using standard integer programming methods, such as dynamic programming, branch and bound, and Lagrangian relaxation. Because we are only concerned with precedence constraints of the simple fork/join type with small N_i, the minimization of (9) is done by enumeration with a simple bounding step. That is, L_i is computed for each possible value of $\{b_{ij}\}$, and $\{b_{ij}^*\}$ are the ones yielding the lowest value. The discussion of the simple bounding step will be omitted here. The complexity for solving the subproblem is related to the time horizon K and the number of operations of the fork/join precedence graph. Thus, the complexity for a job with three operations is on the order of K^3, which is designated $\mathcal{O}(K^3)$. As mentioned above, this subproblem could be solved with Lagrangian relaxation, and, in fact, this is the subject of our current investigation.

Solving the Dual Problem

To solve the dual problem of (7), several methods for generating the dual solution π^* have been presented recently (see, e.g., [21] for a multiplier adjustment method, [22] for a column generation procedure, and [23] for a subgradient optimization procedure). We adopt the subgradient method used in our previous papers [9], [10]. It was originally presented by [24], further explored by [25], and commonly used to solve this type of problem (see, e.g., [15], [16], and [19]).

In our algorithm, the multiplier π^n at iteration n changes according to the step size α^n and the subgradient $g(\pi^n)$ of L with respect to π.

$$\pi^{n+1} = \pi^n + \alpha^n g(\pi^n) \qquad (10)$$

The kth component of the subgradient is given by the kth capacity constraint (3) according to (5).

$$g_k(\pi) = \sum_i \delta_{ik} - M_k \qquad (11)$$

The step size α^n is a function of the difference between the value of L at the nth iteration and an estimate $\bar{L}$ of the optimal value, the inner product of the gradient, and a parameter λ.

$$\alpha^n = \lambda(\bar{L} - L^n)/[g(\pi^n)^T g(\pi^n)]$$
$$\text{for } 0 < \lambda < 2 \qquad (12)$$

Using (12) as the step size, this method converges at a geometric rate (order 1) [25]. An adaptive step-sizing mechanism is used here. It is a modified version of that suggested by [22], and it also incorporates features used in [19], with parameters selected based on testing experiences. The subgradient algorithm terminates either when α^n remains small for a fixed number of iterations while L^n is nonincreasing or when a fixed number of iterations has been reached.

Construction of a Feasible Schedule

Because of the discrete decision variables involved and the stopping criterion used, the solution to the dual problem is generally associated with an infeasible schedule, i.e., capacity constraints (3) might be violated for a few time slots. Note that precedence constraints (2) and processing time requirements (4) are always satisfied in view of how the subproblems of (9) are solved. To construct a feasible schedule, a heuristic approach based on the list-scheduling concept is developed as follows. From the dual solution, each operation is uniquely associated with a beginning time, b_{ij}^*. A list is created by arranging operations of all jobs in the ascending order of b_{ij}^*. The operations are then scheduled on machines according to this list as machines become available, subject to precedence constraints and machine availability for the remaining processing period of the operation.

If the capacity constraint is violated at time k, a greedy heuristic determines which *new* operations should begin at that time slot and which ones are to be delayed by one time unit. In this greedy heuristic, the incremental change in cost if a new operation is delayed by one time unit is calculated. Operations are then assigned to machines in the descending order of the impact (change in cost), subject to machine availability for the remaining processing period of an operation. When all the machines available at that time slot are assigned, the leftover operations are delayed by one time unit. Subsequent operations of those delayed ones are then checked to determine if precedence constraints are violated. If they are, those operations are also delayed by one time unit. The process then repeats. The pseudocode of this heuristic is provided in the Appendix.

Evaluation of the Feasible Solution via the Approximate Duality Gap

Once a feasible schedule is obtained, the corresponding value of the objective function J is an upper bound on the optimal objective J^*. The value of the dual function L^*, on the other hand, is a lower bound on J^* [17]. The difference between J^* and L^* is known as the duality gap. An upper bound of the duality gap is provided by $J - L^*$, which is a measure of suboptimality of the feasible schedule. To obtain an optimal solution, the branch-and-bound technique can be applied in conjunction with the upper/lower bounds obtained above. However, such a step has exponential computational requirements in the worst case [27]. Furthermore, this step is not justified, as our computational experience shows that the relative approximate duality gap $(J - L^*)/L^*$ is usually very small (less than 1 percent for most tests; see next section).

Test Results

Example 1

In the first example, there are four machines and 11 jobs with a total of 18 operations. Jobs have three different weights (w_i = 16, 9, 1), and the planning horizon is 30 days (i.e., K = 30, and time unit = day). Note that a job with weight 1 must be at least 4 days late to compete for a machine with a job with weight 16 and 1 day late. Data are shown in Table 1. The due date D_i is interpreted as the number of days from the current day (time 0). Thus, the due date −2 of job 3 means that the job was due 2 days ago. The smallest arrival time A_i for all jobs is 1; i.e., we are preparing the schedule for the next day. Also, only the first two machines are available at the outset. The third and fourth machines are available starting on day 2.

The value of the optimal dual solution is 230.5. The primal solution at this point is, as expected, infeasible. Following the heuristic procedure outlined above, a feasible schedule is constructed, as represented by the Gantt chart of Table 2. The value of the feasible schedule is 231.00, a relative difference of 0.217 percent compared to the value of the dual solution. Since all the weights are integers, the tardiness penalty should be an integer number. Therefore, the schedule represented in Table 2 is actually optimal. The cost is composed of tardiness due to job 3 with weight 9 and due date −2, and jobs 5, 8, and 11 all with weight 1. Note that job 3 is started immediately and completed as quickly as possible. The high degree of parallelism in the data models the day shift/night shift situation where M1 and M3 are on day shift and M2 and M4 are on night shift. The time to solve the problem was 4.9 CPU sec on an IBM 3090 mainframe computer with the initial value of π_k equal to 0 for all k.

Example 2

In the second example, there are 13 machines and 53 jobs: six jobs have weight 16; 14 jobs have weight 9; and 33 jobs have weight 1. These 53 jobs have a total of 100 operations among them, an average of about two operations per job and eight operations per machine. The planning horizon is 50 days with the time unit measured in days.

The value of the dual solution is 3045.00. The feasible schedule has a cost of 3045.00 and is optimal, as well. The optimal schedule is characterized by a total of seven late jobs, where five of them start on day 1. Exact data and the resulting schedule are available upon request. This problem was solved in 7.2 CPU sec on an IBM 3090 with the initial value of π_k equal to 0 for all k. It can be seen that the algorithm is not only effective but also efficient.

Example 3

For the third example, there are 44 machines and 112 jobs with a total of 210 operations. Due dates are assumed to be poorly selected, so that most of the jobs are now past due. There are five weights: two jobs have weight 16; 19 jobs have weight 9; three jobs have weight 4; 84 jobs have weight 1; and four jobs have weight 0.5. The planning horizon extends 247 days, almost a year of working days (excluding weekends and holidays).

Table 1 Data for Example 1

Job i	w_i	A_i	D_i	Op. j	I_{ij}*	t_{ij}	s_{ij}
1	1	3	5	1	—	1	—
2	1	1	2	1	—	1	—
3	9	1	−2	1	2, 3	1	1
				2	—	1	—
				3	—	1	—
4	9	1	13	1	—	1	—
				2	—	1	—
5	1	3	5	1	—	2	—
6	16	1	5	1	—	3	—
				2	—	3	—
7	16	1	13	1	—	5	—
				2	—	5	—
8	1	1	15	1	3, 4	4	7
				2	3, 4	4	7
				3	—	3	—
				4	—	3	—
9	9	1	23	1	3	3	3
				2	3	3	3
				3	—	1	—
10	9	1	3	1	—	1	—
				2	—	1	—
11	1	1	4	1	—	2	—

*The set I_{ij} consists of all operations immediately following operation (i, j).

Table 2 Gantt Chart for Example 1

Time	1	2	3	4	5	6	7	8	9	10
M1	2, 1	6, 1	6, 1	6, 1	8, 2	8, 2	8, 2	8, 2	4, 2	9, 2
M2	3, 1	6, 2	6, 2	6, 2	8, 1	8, 1	8, 1	8, 1	4, 1	9, 1
M3	N/A	10, 1	3, 2	1, 1	5, 1	5, 1*	7, 2	7, 2	7, 2	7, 2
M4	N/A	10, 2	3, 3*	11, 1	11, 1*	7, 1	7, 1	7, 1	7, 1	7, 1

Time	11	12	13	14	15	16	17	18	19	20
M1	9, 2	9, 2			8, 3	8, 3	8, 3			
M2	9, 1	9, 1			8, 4	8, 4	8, 4*			
M3						9, 3				
M4										

*Late job.

The lower bound on the optimal schedule is 1,018,130.8, while the feasible schedule obtains a cost equal to 1,018,432.5, a difference of 0.03 percent. Almost all the jobs are late, most of them more than 20 days late. Exact data and the resulting schedule are available upon request. The schedule was obtained in 17.0 CPU sec on an IBM 3090 mainframe with the initial value of π_k equal to 0 for all k.

What-If Studies and Schedule Reconfiguration

Job-Interaction Parameters

For convex programming problems, it is known that an optimized Lagrange multiplier is the sensitivity of the cost function with respect to the level of the corresponding constraint [17]. That is, suppose that M_k and b_{ij} are continuous variables instead of integers. Then π_k^* is the sensitivity of the optimal cost J^* with respect to the level of the constrained resource M_k, as represented by the derivative of J^* with respect to the resource level at time k.

$$\pi_k^* = -dJ^*/dM_k \tag{13}$$

For our problem, machines are available in discrete units, and beginning times are also discrete variables. As a result, the derivative defined above does not exist, and π_k^* is only an estimate of the sensitivity of J^* with respect to M_k.

$$\pi_k^* \approx -\Delta J^*/\Delta M_k \tag{14}$$

Another well-established property of the optimized Lagrange multiplier π_k^* is that it is the "price" of using a machine at time k, the so-called "shadow price" in economic terminology. Therefore, the cost of job i can be viewed as the sum of the penalty for missing the due date and the cost of using a machine. The cost of using a machine at time k is higher if many jobs compete for machines at that time to meet their respective due dates. Consequently, the optimized Lagrange multipliers are also a measure of the competition among jobs for machines. For this reason, we have referred to the optimized Lagrange multipliers as "job-interaction variables."

The above properties of Lagrange multipliers have been exploited in [10] for single-operation jobs to measure the impact of a wide variety of dynamic changes, which may occur within the time horizon. These ideas will be extended here to estimate the effect of longer processing time and adding a new operation to an existing job. Since the evaluation of these estimates takes much less time than rerunning the algorithm, a scheduler can quickly determine the relative impact of a number of changes without actually reconfiguring the schedule. This exploration of options has been referred to as a "what-if" study. The scheduler can then implement any particular changes (including the addition of new jobs and deletion of completed jobs) via reconfiguration of the existing schedule. Since changes in jobs and machines are usually "small" compared to the total amount of jobs and machines under consideration, Lagrange multipliers can be easily updated. Reconfiguration can, therefore, be efficiently done on a daily basis to incorporate new and/or completed jobs and to accommodate other dynamic changes in the system.

Effect of Longer Processing Times

Suppose operation (i, j) requires 1 day longer than originally anticipated. This is equivalent to the situation where a machine becomes unavailable or breaks down at time $c_{ij} + 1$. In addition to machine unavailability, subsequent operations of job i may have to be delayed by 1 day and the job tardiness penalty might be increased. Denoting by $\bar{T}_i$ the estimated new tardiness, an estimate of the change is given by adding the increased tardiness penalty together with a sum of Lagrange multipliers (or prices for machine usage) at times associated with violated precedence constraints (represented below by l).

$$\bar{J} \approx J + w_i(\bar{T}_i^2 - T_i^2) + \pi^*_{c_{ij}+1} + \sum_l (\pi^*_{c_{il}+1} - \pi^*_{b_{il}}) \tag{15}$$

The expression in (15) can be easily extended to the general case where (i, j) requires many more days than originally anticipated. Clearly, the estimate $\bar{J}$ can be obtained almost instantaneously. The schedule can also be efficiently and effectively reconfigured after multipliers are updated. Similar analysis holds for the case where an operation finishes earlier than anticipated.

Example 4: Changes in Processing Times

Suppose, in Example 2, an operation of a job with weight 9 requires three less days to process, a change from 5 to 2 days. The estimated change in cost is 2585.55, a 15.09 percent decrease from the current schedule calculated in almost zero CPU time. Multipliers are then updated by using the subgradient algorithm, and the resulting schedule has a cost of 2586 obtained in 10.2 CPU sec.

Now suppose a job with weight 16 requires two more days to process, from 1 day to 3 days. All other things are the same as in Example 2. The estimated new cost is 3493.22, a 14.72 percent increase and acquired in almost no CPU time. This compares favorably with the cost of 3493 after reconfiguration, obtained in 9.3 CPU sec.

Effect of Scheduling a New Operation of an Existing Job

Suppose we add a new operation (i, q) between an existing pair of consecutive operations (i, p) and (i, r) of job i. For simplicity, we assume that operation (i, q) has processing time $t_{iq} = 1$ and time-out $s_{iq} = 0$. Inserting operation (i, q) into the schedule will delay the completion time of job i and other existing jobs. If operation (i, q) is scheduled before the s_{ip} time-out, this problem reduces to that discussed above with the $c_{ij} + 1$ subscript of (15) replaced by $c_{ip} + 1$. If operation (i, q) is to be scheduled after the s_{ip} time-out, the expression (15) is also valid with $c_{ij} + 1$ replaced by $c_{ir} + 1$. Again, the extension to longer operations and with time-outs is straightforward and shall be omitted here. The schedule can also be reconfigured after multipliers are updated.

Example 5: Scheduling a New Operation of an Existing Job

Again, using the data from Example 2, a new operation is inserted into a job directly after another operation with the following characteristics: Job due date is 1 day from the current day, job weight is 9, and the new processing time is 3 days with no additional slack time. By extending (15), the estimated new cost is 3668.37, a 20.47 percent increase from the original schedule. The reconfigured cost is 3666, obtained in 11.0 CPU sec, with the lower bound being 3662.83.

Scheduling New Jobs

Most scheduling situations are not static; not only are processing times and process plans changed, but existing jobs are completed and new jobs arrive. For some scheduling methodologies—e.g., branch and bound—the schedule has to be regenerated from scratch to accommodate completed jobs, new jobs, or any other dynamic changes. For our method, completed jobs and new jobs are handled by rerunning the subgradient algorithm. Since the total number of jobs and their overall characteristics generally do not change much from one day to another, one day's multipliers can be easily updated to provide a near-optimal schedule for the next day, allowing more efficient regeneration of the schedule.

Example 6: Scheduling New Jobs

We use the data of Example 3 to demonstrate the advantage of retaining multiplier values in daily scheduling operations. Recall that the schedule of Example 3 was obtained in 17.0 CPU sec with π_k initialized at 0 for all k. Suppose the next day (day 1), six jobs are completed and five jobs arrive, resulting in a total of 111 jobs and 202 operations. Using the multipliers obtained in Example 3, we obtain a solution with $(J - L^*)/L^*$ equal to 0.07 percent in 7.1 CPU sec. Starting with zero multipliers, the solution is obtained in 10 sec. Longer CPU times have been observed if one starts with zero multipliers for other test data.

Now suppose the following day (day 2), seven jobs are completed and five more jobs arrive, resulting in a total of 109 jobs with 198 operations. Again, a solution is obtained with $(J - L^*)/L^*$ equal to 0.08 percent in 5.0 CPU sec when we start with day 1's multipliers. This is in contrast to the 11.4 CPU sec required if one starts with zero multipliers.

Conclusions

This article presents a methodology for scheduling jobs with simple precedence constraints and due dates on identical parallel machines. Time-outs for visits to other machines or offices are also considered. The special integer programming formulation of the problem facilitates the application of the Lagrangian relaxation technique. Decomposition of the dual problem serves to simplify the solution at the low level. The high-level problem is then solved via a subgradient method. A heuristic using the list-scheduling concept is developed to construct a feasible solution based on the dual results. The complexity of the solution methodology is related to the number of operations in the precedence structure. For most of the problems tested, results are within 1 percent of the lower bounds and obtained within practical amounts of CPU time. More importantly, the interaction of jobs as they compete for limited resources becomes visible. By using the job-interaction information, the method can provide quick answers to what-if questions, allow reconfiguration of an existing schedule when changes occur, and also accommodate new jobs.

This problem is grounded in the synchronous manufacturing philosophy, where bottlenecks are deemed to control the flow of jobs through a shop. The methodology is amenable to large-scale problems and is oriented toward a dynamic view of the manufacturing environment. Furthermore, the what-if feature is designed to help schedulers choose among the options that are under their control. These capabilities have significant value for Pratt & Whitney engineers in their scheduling operations. We believe that this approach will have further impact when extended to a more general production setting.

Appendix: Pseudocode of the Heuristic Procedure

Define S_j as a sequence of $P = \Sigma_i N_i - j + 1$ operations $[i_1, 1], [i_2, 2], \ldots, [iP, P]$, where the first subscript designates the job of which the operation is a part and, thus, may not be unique. Associated with this set of P operations is a set of P beginning times $\{b[i_l, l]\}$, ordered from the smallest to the largest so that $b[i_l, l] \leq b[i_{l+1}, l + 1]$ for $l = 1, 2, \ldots, P$. If $b[i_l, l] = b[i_{l+1}, l + 1]$, then $f(l) > f(l + 1)$ where $f(l)$ is the incremental change in the cost function, defined as the change in tardiness penalty when the operation is scheduled one day later.

$$f(l) = w_l[(T_l + 1)^2 - T_l^2] \quad \text{(A1)}$$

In addition, define j as the operation index and n as a time-index tracking machine availability. Let k represent a time index that allows an operation to start between n and its scheduled beginning time if the precedence constraints are not violated. Let E be the set of jobs that cannot be scheduled between time n and their respective beginning times $b[i_j, j]$. Given the sequence S_1 and $\{M_k\}$, the greedy heuristic algorithm works as follows:

Step 0: [*Initialization.*] Set $j = 1$ and go to Step 1.

Step 1: [*Increment Iterations.*] Determine the first time l such that $M_l > 0$ and set $n = l$, $k = n$, and $E = \varnothing$ and go to Step 2.

Step 2: [*Check Capacity at Time k.*] If $M_l \neq 0$ for $l = k, k + 1, k + t[i_j, j] - 1$, go to Step 3; otherwise go to Step 4.

Step 3: [*Assign* $[i_j, j]$ *and Modify Machine Availability.*] If the precedence constraint is not violated, schedule operation $[i_j, j]$ to begin at time k, set $M_l = M_l - 1$ for $l = k, k + 1, \ldots, k + t[i_j, j] - 1$ and go to Step 7; otherwise go to Step 4.

Step 4: [*Increment Time k.*] Set $k = k + 1$. If $k > b[i_j, j]$, go to Step 5; otherwise go to Step 2.

Step 5: [*Check Operations to be Scheduled at the Same Time Slot.*] If $b[i_{j+1}, j + 1] = b[i_j, j]$, set sequence $S_j = S_j - \{[i_j, j]\}$, reindex the sequence, and set $E = E \cup \{[i_j, j]\}$, $k = n$ and go to Step 2; otherwise go to Step 6.

Step 6: [*Update Sequence* S_j.] Set $S_j = S_j \cup E$; for any unscheduled operations, if $b[i_l, l] < k$: Reset $b[i_l, l] = k$, check all subsequent operations of job $[i_l]$ and reset those beginning times that violate precedence constraints; reform sequence S_j and go to Step 1.

Step 7: [*Stopping Criterion.*] Set $j = j + 1$; if $j > \Sigma_i N_i$, stop; otherwise go back to Step 2.

Acknowledgments

This work was supported, in part, by the National Science Foundation under Grants ECS-8513163 and ECS-8717167. The authors would like to thank Scott Bailey, Jack Gibbs, Steve Cochran, Curtis Cook, and Richard Lopatka of Pratt & Whitney for their invaluable suggestions and support.

References

[1] E. M. Goldratt and J. Cox, *The Goal: A Process of Ongoing Improvement*, Croton-on-Hudson, NY: North River Press, 1986.

[2] J. K. Lenstra and A. H. G. Rinnooy Kan, "Complexity of Scheduling under Precedence Constraints," *Operations Research*, vol. 26, no. 1, pp. 22-35, Jan.-Feb. 1978.

[3] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Recent Developments in Deterministic Sequencing and Scheduling: A Survey," in *Deterministic and Stochastic Scheduling*, M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan, eds., Boston: D. Reidel Publishing Co., 1982.

[4] S. C. Graves, "A Review of Production Scheduling," *Operations Research*, vol. 18, pp. 841-852, 1981.

[5] M. R. Garey, R. L. Graham, and D. S. Johnson, "Performance Guarantees for Scheduling Algorithms," *Operations Research*, vol. 26, pp. 3-21, 1978.

[6] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM J. Appl. Math.*, vol. 17, pp. 416-429, 1969.

[7] D. B. Fogel, "Addressing the Traveling Salesman Problem Through Evolutionary Adaptation," *Biological Cybernetics*, vol. 60, no. 2, pp. 139-144, 1988.

[8] S. Kirkpatrick, C. D. Gelatt, and C. M. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, 1983.

[9] P. Luh, D. Hoitomt, K. Pattipati, and E. Max, "Parallel Machine Scheduling Using Lagrangian Relaxation," *Proc. Int. Conf. Comp. Integr. Manuf.*, pp. 244-248, Troy, NY, May 1988.

[10] P. Luh, D. Hoitomt, K. Pattipati, and E. Max, "Schedule Generation and Reconfiguration for Parallel Machines," *Proc. 1989 IEEE Int. Conf. Robot. and Automa.*, pp. 528-533, Scottsdale, AZ, May 1989.

[11] J. N. Lin and Y. P. Zheng, "Development of a Multilayer Hierarchical Scheduling Al-

gorithm for Manufacturing Systems with Identical Machines," *Proc. 1989 American Cont. Conf.*, pp. 2400-2404, Pittsburgh, PA, June 1989.

[12] N. T. Bruvold and J. R. Evans, "Flexible Mixed-Integer Programming Formulations for Production Scheduling Problems," *IIE Trans.*, vol. 17, no. 1, pp. 2-7, Mar. 1985.

[13] M. I. Norbis and J. M. Smith, "Two-Level Heuristic for the Resource Constrained Scheduling Problem," *Int. J. Production Research*, vol. 24, pp. 1203-1219, 1986.

[14] Everett, H., "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," *Operations Research*, vol. 11, pp. 399-417, 1963.

[15] M. L. Fisher, "Optimal Solution of Scheduling Problems Using Lagrange Multipliers, Part I," *Operations Research*, vol. 21, pp. 1114-1127, 1973.

[16] R. Conterno and Y. C. Ho, "Order Scheduling Problem in Manufacturing Systems," *Int. J. Production Research*, vol. 26, no. 9, pp. 1487-1510, Sept. 1988.

[17] D. G. Luenberger, *Linear and Nonlinear Programming*, 2nd Ed., Reading, MA: Addison-Wesley, 1984.

[18] A. M. Geoffrion, "Lagrangean Relaxation for Integer Programming," *Mathematical Programming Study*, vol. 2, pp. 82-114, 1974.

[19] N. R. Sandell, Jr., D. P. Bertsekas, J. J. Shaw, S. W. Gully, and R. F. Gendron, "Optimal Scheduling of Large-Scale Hydrothermal Power Systems," *Proc. 1982 IEEE Int. Large-Scale Systems Symp.*, pp. 141-147, Virginia Beach, VA, Oct. 1982.

[20] B. Gavish and I. Neuman, "A System for Routing and Capacity Assignment in Computer Communication Networks," *IEEE Trans. Communications*, vol. 37, no. 4, pp. 360-366, Apr. 1989.

[21] M. Held and R. M. Karp, "The Traveling Salesman Problem and Minimum Spanning Trees: Part II," *Mathematical Programming Study*, no. 1, pp. 6-25, 1971.

[22] M. L. Fisher, "Lagrangian Relaxation Method for Solving Integer Programming Problems," *Management Science*, vol. 27, pp. 1-18, 1981.

[23] B. Gavish and S. K. Srikanth, "Optimal Solution Methods for Large-Scale Multiple Salesmen Traveling Salesman Problem," *Operations Research*, vol. 34, pp. 698-717, 1987.

[24] N. Z. Shor, "On the Structure of Algorithms for the Numerical Solution of Optimal Planning and Design Problems," *Diss. kand. fiz.-matem. n.*, Kiev, In-t kiberbetika AN USSR, 1964.

[25] B. T. Polyak, "Minimization of Unsmooth Functionals," *USSR Computational Math. and Math. Physics*, vol. 9, pp. 14-29, 1969.

[26] K. R. Pattipati, J. J. Shaw, J. C. Deckert, L. K. Beean, M. G. Alexandridis, and W. P. Lougee, "CONFIDANTE: A Computer-Based Design Aid for the Optimal Synthesis, Analysis and Operation of Maintenance Facilities," *Proc. 1984 IEEE AUTOTESTCON*, Nov. 1984.

[27] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Rockville, MD: Computer Science Press, p. 575, 1978.

Debra J. Hoitomt received the B.S. degree in education in natural science from the University of Wisconsin, Madison, in 1977, and the M.S. degree in systems engineering from the University of Arizona, Tucson, in 1984. In 1987, she joined the Manufacturing Information Technology group at Pratt & Whitney, East Hartford, to become part of an effort to develop a large-scale, optimization-based, distributed scheduling methodology. She is currently a doctoral candidate at the Department of Electrical and Systems Engineeering, University of Connecticut, Storrs. Her current research interests are analysis and control of production systems, constrained optimization, numerical methods, and distributed processing.

Peter B. Luh received his B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, Republic of China, in 1973, the M.S. degree in aeronautics and astronautics engineering from MIT, Cambridge, Massachusetts, in 1977, and the Ph.D. degree in applied mathematics from Harvard University, Cambridge, Massachusetts, in 1980. Since 1980, he has been with the University of Connecticut, and currently is an Associate Professor in the Department of Electrical and Systems Engineering. His major research interests include hierarchical planning and control of large-scale systems, schedule generation and reconfiguration for manufacturing systems, distributed decision making, game theory, and multitarget tracking. He has been a Principal Investigator and Consultant to many industry- and government-funded projects in the above areas, and he has published about 80 papers. He is an Associate Editor of the *IEEE Transactions on Automatic Control*, won the Best Paper Award of the 1987 Joint Command and Control Research Symposium, and has served on Program Committees and Operating Committees of several major national, international, and intersociety conferences. He is a Member of the IEEE; he is also a member of Sigma Xi and is listed in *Who's Who in Engineering* and *Who's Who in the East.*

Eric Max is Project Leader at Pratt & Whitney's development shop, where he saw a need to increase the accuracy of projected job completion dates and provide tools enabling more informed job priority decisions. He formed an interdepartmental project team combining the disciplines of production planning and control, artificial intelligence, mathematical optimization, and common sense. Under his guidance, and with academic support, a distributed scheduling methodology developed, which works within the natural decision-making process inherent in a multilayer organizational structure. Mr. Max received his B.S. from Cornell University and his M.B.A. from Boston University; he is currently the Inventory Control Manager at Pratt's Overhaul and Repair Center.

Krishna R. Pattipati received the B.Tech. degree in electrical engineering with highest honors from the Indian Institute of Technology, Kharagpur, in 1975, and the M.S. and Ph.D. degrees in systems engineering from the University of Connecticut in 1977 and 1980, respectively. He was employed by ALPHATECH, Inc., Burlington, Massachusetts, from 1980 to 1986, where he supervised and performed research on human decision modeling in large-scale air defense systems and power networks, multitarget tracking, queuing-network-based performance evaluation of fault-tolerant computer architectures, automated testing, and large-scale mixed-integer optimization. Since September 1986, he has been an Associate Professor in the Department of Electrical and Systems Engineering. Dr. Pattipati was selected by the IEEE Systems, Man, and Cybernetics Society as the Outstanding Young Engineer of 1984 and received the Centennial Key to the Future award. He won the Best Technical Paper Award at the 1985 IEEE AUTOTEST Conference. He was a member of the Administrative Committee of the IEEE Systems, Man, and Cybernetics Society during 1986-1988. Dr. Pattipati served as the Finance Chairman of the International Conference on Control and Applications, Jerusalem, Israel, in 1989 and as a Vice Chairman for invited sessions of the IEEE International Conference on Systems, Man, and Cybernetics, Boston, in 1989. He is listed in *Who's Who in the Frontiers of Science and Technology.*