A Lagrangian Relaxation Method for Parallel Machine Scheduling with Resource Constraints

Daecheol Kim*

〈Abstract〉

This research considers the problem of scheduling jobs on parallel machines with non-common due dates and additional resource constraints. The objective is to minimize the total absolute deviation of job completion times about the due dates. Job processing times are assumed to be the same. This problem is motivated by restrictions that occur in the handling and processing of jobs in certain phases of semiconductor manufacturing and other production systems. We examine two problems. For the first of these, the number of different types of additional resources and resource requirements per job are arbitrary. The problem is formulated as a zero-one integer linear programming and the Lagrangian relaxation approach is used. For the second case, there exists one single type of additional resource and the resource requirements per job are zero or one. We show how to formulate the problem as an assignment problem.

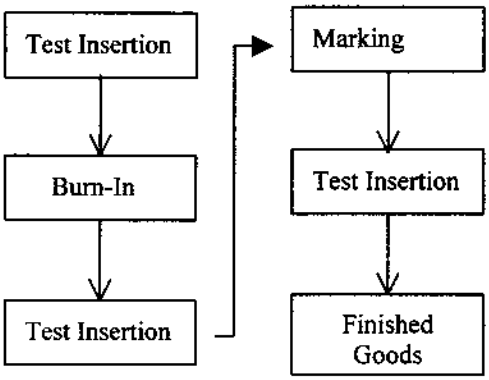
1. Introduction

The study of parallel machine scheduling with additional resource constraints is a new area of research. In most of the classical parallel machine scheduling problems only machines are constrained resources [1]. Thus, at any time, the execution of a job is restricted by the availability of a single scarce resource. However, in some manufacturing systems, jobs may also require, besides machines, certain additional limited resources for their handling and processing, such as automated guided vehicles, machine operators, tools, pallets, fixtures, industrial robots, etc. A practical example of such a scheduling problem can be found in burn-in operations in the final testing stage of semiconductor manufacturing. Very large-scale integrated (VLSI) circuits are manufactured by the process which ca be divided into four basic steps: wafer fabrication, wafer probe test, packaging, and final testing. The goal of the testing process is to determine whether or not each integrated circuit is operating at the required specification. Product flows through the test area in lots. While there is considerable variety in process flows, a general idea of product flow can be formed from Figure 1. In the burn-in operation, VLSI circuits undergo thermal stress in the ovens for a certain period time at a constant temperature (generally about 120°C) in order to sort out latent defects. When a lot of circuits arrive at the burn-in area, the circuits are loaded onto boards, and the absence of the necessary board prevents the lot from being loaded into the ovens. In practice this corresponds to assigning the available boards to the lots awaiting burn-in off-line and treating each board as an individual job. Thus, boards are considered as additional limited resources. Once the boards are loaded into the ovens, no board can be removed from the machine until the processing of the job is completed [2][3].

There have appeared in the literature a few scheduling models, which explicitly take additional limited resources into consideration [4]. Garey et al. examine the computational

* LG-EDS

complexity of multiprocessor scheduling problems, where the scheduling length is the criterion [5]. In [6], models involving parallel machines, unit processing time jobs and the maximum lateness criterion are studied in detail. Blazewicz et al. consider scheduling of independent jobs on an arbitrary number of identical parallel processors to minimize mean flow time, with one resource type and zero-one resource requirements. They prove that an optimal schedule exists such that all jobs requiring a unit of additional resource available in b units can be assigned to the first b machines [7].



〈Figure 1〉 Example product flow

Most previous parallel machine scheduling research with additional resources has dealt with performance criterion such as makespan, mean tardiness, mean lateness, etc. With the increasing interest in JIT production, the emphasis has shifted towards earliness and tardiness (E-T) scheduling taking earliness as well as tardiness into consideration. The objective function considered for this problem is consistent with the JIT production philosophy which espouses the notion that earliness as well as tardiness should be penalized, i.e., jobs that complete earlier than their due dates tie up capital, increase inventory levels, and take up scarce floor space, while jobs that complete after the due dates may cause a customer to shut down operations. Even if there are a number of different ways to represent the JIT concept in scheduling models, an important special case in the family of E-T problems involves minimizing the total absolute deviations of job completion times from the due dates [8].

The problems studied in this research are defined as follows. We are given a set of n jobs $J=\{1,2,...,n\}$ to be processed on a set of m identical parallel machines $M=\{M_1,M_2,...,M_m\}$. p_j and r_j are processing and ready times of job j, respectively. Each job j has its distinct due date, d_j. Job processing times are assumed to be the same (i.e., $p_j=1$). In this paper, two problems are addressed. For the first of these (P1), there exist u types of additional resources $R_1,R_2,...,R_u$ available in $b_1,b_2,...,b_u$ units, respectively. Each job j requires for its processing one machine and certain amounts of additional resources specified by the resource requirement vector $R(j) = [R_1(j), R_2(j),...,R_u(j)]$, where $R_k(j)$ $(0 \le R_k(j) \le b_k)$, k=1,2,...,u, denotes the number of units of resource type k required for the processing of job j. For the second problem (P2), there exists one single type of additional resource (i.e., u = 1) and the resource requirements per job are zero or one (i.e., R(j) = 0 or 1). We assume here that all the required resources are allocated to a job when its processing begins, and they are returned by the job right after its completion. Resources are referred to as renewable under this assumption [9]. Without loss of generality we assume that all p_j, d_j, r_j, $R_k(j)$, and b_k are integers.

The rest of the paper is organized as follows. In Sections 2-5, the first problem is analyzed. Section 2 provides a zero-one integer linear programming, and the associated dual formulation is also discussed. In Section 3, two heuristic procedures to find an initial feasible solution and to transform an infeasible solution to a feasible one are presented. In Sections 4 and 5, subgradient optimization procedure and implementation issues are discussed. In Section 6, computational results are also provided. In Section 7, it is shown that the second problem is reduced to an assignment problem. Section 8 presents conclusion and future research.

2. Formulation of P1

This problem is the case in which the number of different types of additional resources is arbitrary, and each job requires an arbitrary number of units of additional resources for each

additional resource type at any point in time. In addition, the ready time and due date of each job are distinct. All processing times are the same, i.e., $p_i=1$, i=1,2,...,n. Let C_{jt} be the cost of job j which completes its processing at t. Then, C_{jt} can be expressed as follows:

$$C_{jt} = \begin{cases} \lambda_1(d_j - t) & \text{if } d_j \geq t, \\ \lambda_2(t - d_j) & \text{if } d_j \leq t, \\ \infty & \text{otherwise,} \end{cases}$$

where λ_1 and λ_2 are weights of earliness and tardiness, respectively.

With E-T as an optimality criterion, the zero-one ILP formulation for problem (P1) can be expressed as:

(P1)

$$\min \sum_{j=1}^{n} \sum_{t=t_1}^{t_h} C_{jt}\, x_{jt}\,, \tag{0}$$

s.t.

$$\sum_{j=1}^{n} x_{jt} \leq m, \qquad t=t_1,t_1+1,...,t_h, \tag{1}$$

$$\sum_{j=1}^{n} R_k(j)x_{jt} \leq b_k, \qquad t=t_1,t_1+1,...,t_h;\ k=1,...,u, \tag{2}$$

$$\sum_{t=t_1}^{t_h} x_{jt} = 1, \qquad j=1,2,...,n, \tag{3}$$

$$x_{jt} \in \{0, 1\}, \qquad j=1,...,n;\ t=t_1,...,t_h, \tag{4}$$

where

$$x_{jt} = \begin{cases} 1 & \text{if job j is completed at time t,} \\ 0 & \text{otherwise.} \end{cases}$$

Constraint (1) ensures that no more than m jobs are assigned to any time unit and (2) guarantees that the resource requirements of all the jobs processed during the time interval (t-1, t) are within the resource limit. Constraints (3) expresses that each job is assigned to only one time.

Lagrangian relaxation is an approach to handle computationally hard integer programming problems. Specifically, some sets of difficult constraints are dualized to create an LRP which is easier to solve. For a minimization problem, the optimal objective function value of LRP is a lower bound to the optimal objective value of the primal [10]. In this problem, constraint set (2) is dualized so that the associated LRP has a special structure and accordingly, can be easily solved. By relaxing constraint set (2), the following transportation problem is obtained:

$$\text{(LRP) } Z_D(w) = \min \sum_{j=1}^{n} \sum_{t=t_1}^{t_h} (C_{jt} + \sum_{k=1}^{u} w_{tk} R_k(j))\, x_{jt} - \sum_{t=t_1}^{t_h} \sum_{k=1}^{u} w_{tk}\, b_k \tag{5}$$

s.t.

$$\sum_{j=1}^{n} x_{jt} \leq m, \qquad t=t_1,t_1+1,...,t_h, \tag{6}$$

$$\sum_{t=t_1}^{t_h} x_{jt} = 1, \qquad j=1,2,...,n, \tag{7}$$

$$x_{jt} \in \{0, 1\}, \qquad j=1,..,n;\ t=t_1,...,t_h, \tag{8}$$

where w_{tk}, $t=t_1,...,t_h, k=1,...,u$, are the non-negative Lagrangian multipliers for constraint set (2). The Lagrangian dual program (LDP) is then to find w* that makes the lower bound as large as possible, i.e.,

(LDP)

max $Z_D(w)$

subject to $w \geq 0$.

3. Heuristic Procedures

To start the Lagrangian relaxation procedure, a good initial primal feasible solution is necessary. Also, while executing the Lagrangian relaxation procedure, the solution to LRP is converted into a feasible solution to the original problem by suitable adjustment if necessary. This feasible solution constitutes an upper bound on the optimal solution to the primal problem. In the following, two heuristic algorithms are described. The first algorithm generates an initial primal feasible solution and the second one attempts to convert the

solution to LRP into a feasible solution for the original problem.

3.1 Heuristic 1 Procedure to Find an Initial Feasible Solution

The basic idea of the first heuristic procedure, called Heuristic 1 (H1), is as follows. Jobs are arranged in nondecreasing order of their due dates. Then, the job with the smallest due date is assigned to the best feasible position. The assignment of job j to a machine at moment t is feasible if the following three conditions are satisfied: (i) $R_{tk} - R_k(j) \geq 0$, k=1,...,u, (ii) $N_t < m$, and (iii) $r_j < t$. R_{tk} denotes the available amount of additional resource of type k at moment t and N_t a counter of jobs assigned at the same moment t. Obviously, the best position for a job is its due date. Thus, the algorithm starts the search at its due date, and then alternately checks around the due date until a feasible position is found. The distance from the position of a job to its due date is stored in a counter, inc. This process is continued for the remaining jobs. Ties are broken arbitrarily. In this algorithm, j denotes the job index and z^r the upper bound obtained from H1. Assume that jobs are indexed by non-decreasing order of their due dates. A complete description of the H1 algorithm is provided below:

(H1)

Step 0: Set j=1, z^0=0, N_t=0, and R_{tk}=b_k, k=1,...,u.

Step 1: Set t=d_j.

At moment t, if the assignment of job j to the first available machine is feasible, then update j=j+1, N_t=N_t+1, and $R_{tk} = R_{tk} - R_k(j)$ for k=1,...,u, and go to Step 4.

Otherwise, set inc = 1 and go to the next step.

Step 2: If the assignment of job j to the first available machine is feasible at moment t = d_j - inc, then update j=j+1, $z^0 = z^0$ + inc, $N_t = N_t + 1$ and $R_{tk} = R_{tk} - R_k(j)$ for k=1,...,u, and go to Step 4. Otherwise, go to the next step.

Step 3: If the assignment of job j to the first available machine is feasible at moment t = d_j + inc, then update j=j+1, $z^0 = z^0$ + inc, $N_t = N_t + 1$ and $R_{tk} = R_{tk} - R_k(j)$ for k=1,...,u, and go to Step 4. Otherwise, set inc = inc + 1 and go to Step 2.

Step 4: If j > n, stop; otherwise, go to Step 1.

3.2 Heuristic 2 Procedure to Transform an Infeasible Solution to a Feasible Solution

The basic idea of the second procedure, called Heuristic 2 (H2), is similar to H1. In H1, the best feasible positions for all jobs are searched around job due dates. The schedule generated by H2 is built on the optimal solution to the current LRP. Thus, if the position of a job obtained from LRP is not feasible, then a search for a feasible position is made around the current position. This process is repeated for each job in the LRP schedule. Let t_l and t_u be the minimum and the maximum job completion times in the LRP schedule. Let $A_t = \{ j \in J \mid x^r_{jt} = 1$, where x^r_{jt} is the solution to the current LRP $\}$, where $t=t_l,t_l+1,...,t_u$. Jobs in A_t are ordered by increasing order of their index. The H2 algorithm generates two schedules. To get the first schedule, the algorithm starts at $t=t_l$, and moves forward to $t=t_u$. At moment t, the algorithm removes the first element (say job j) from A_t and checks to see if the assignment of job j to a machine at moment t is feasible. The assignment is feasible if the following three conditions are satisfied: (i) $R_{tk} - R_k(j) \geq 0$, k=1,...,u, (ii) $N_t < m$, and (iii) $r_j < t$. If it is not feasible, then a search around t is performed until the best feasible position is found. The distance from t is stored in a counter, inc. This process is repeated until $A_t = \phi$. When $A_t = \phi$, the process is repeated with t=t+1 until every job is assigned. The second schedule is generated in a similar manner except that the algorithm moves backward starting at $t=t_u$.

In the following, a detailed description of the H2 algorithm is presented. In this algorithm, t and i denotes a time counter and N_t a counter of jobs assigned at the same moment t. N denotes the total number of jobs assigned so far. Let z^r be the upper bound obtained from H2.

(H2)

Step 0: Set $z^r = 0$, $N = 0$, $N_t = 0$, and $R_{ik} = b_k$, k=1,...,u, FIRST = t, and NEXT = 1.

Step 1: Set t = FIRST.

Step 2: At moment t, let job j be the first element in set A_t. Remove job j from set A_t. If the assignment of job j to the first available machine is feasible, then update $z^r = z^r + |c_j - t|$, $N = N + 1$, $N_t = N_t + 1$ and $R_{ik} = R_{ik} - R_k(j)$ for k=1,...,u, and then go to Step 5. Otherwise, set inc = 1 and i = t and go to the next step.

Step 3: If the assignment of job j to the first available machine is feasible at moment i = t - inc, then update $z^r = z^r + |c_j - i|$, $N = N + 1$, $N_i = N_i + 1$ and $R_{ik} = R_{ik} - R_k(j)$ for k = 1, . . ., u, and go to Step 5. Otherwise, go to the next step.

Step 4: If the assignment of job j to the first available machine is feasible at moment i = t + inc, then update $z^r = z^r + |c_j - i|$, $N = N + 1$, $N_i = N_i + 1$ and $R_{ik} = R_{ik} - R_k(j)$ for k=1,...,u, and go to Step 5. Otherwise, update inc = inc + 1 and go to Step 3.

Step 5: If $A_t \neq \phi$, then go to Step 2. If $A_t = \phi$ and $N < n$, update t = t + NEXT and go to Step 2; otherwise, the first schedule is completed, and go to Step 6.

Step 6: Run the algorithm over again from Step 0 with FIRST = t_k and NEXT = -1 to generate the second schedule. After the second schedule is obtained, select the best schedule and stop.

4. Subgradient Procedure

Subgradient optimization is used to maximize the lower bound obtained from LRP. Subgradient optimization is an efficient computational procedure for solving many classes of linear and nonlinear integer programming problems with linear constraints [11][12]. The behavior of a subgradient algorithm in a variety of combinatorial problems was studied by [13]. A theoretical analysis of the convergence of subgradient algorithms is given by [14].

The subgradient algorithm for LRP includes the following steps:

1. Set initial values for the Lagrangian multipliers, bounds and iteration counter:
   $w^1 = 0$; $LB^0 = 0$; $UB^0 = z^0$; r = 1;
   where z^0 is the objective value of the initial primal feasible solution obtained by H1.
2. Solve LRP with the current value of w^r. Let x^r be the solution and $Z_D(w^r)$ the optimal objective function value of LRP, respectively.
3. Update the lower bound:
   $LB^r = \max \{ LB^{r-1}, Z_D(w^r) \}$.
4. If x^r is feasible to ILP, set $\underline{x}^r = x^r$. Otherwise, generate a feasible solution $\underline{x}^r$ with objective function value $Z(\underline{x}^r)$ for ILP from H2.
5. Update the upper bound:
   $UB^r = \min \{ UB^{r-1}, Z(\underline{x}^r) \}$
6. If $(UB^r - LB^r) / LB^r \leq \varepsilon$ (ε is a small tolerance), stop. Otherwise, go to Step 7.
7. Calculate the subgradients:
   $G^r_{tk} = \sum R_k(j) x^r_{jt} / b_k - 1$ t=1,2,...,t_k; k=1,2,...,u.
8. Compute the step size T^r :

$$T^r = \frac{p^r (UB^r - LB^r)}{\sum_{t=1}^{t_k} \sum_{k=1}^{u} G^{r\,2}_{tk}}$$

   where $0 < \pi^r \leq 2$.

9. Update the Lagrange multipliers:

$$w^{r+1}_{tk} = \max \{0, w^r_{tk} + T^r G^r_{tk} \} \quad t=1,2,...,t_k; \; k=1,2,...,u.$$

10. Update the iteration counter ($r \leftarrow r+1$), and go to Step 2.

The sequence $\{Z_D(w^r)\}$ converges to a minimum if $\lim_{r\to\infty} T^r = 0$ and $\sum_{r=\infty} T^r = \infty$ [13]. Therefore, the values of π^r have to be set at each iteration so that the above conditions are satisfied. After a preliminary computational test, the best

convergence criteria is to set π^1=1.1. Then, if UB^r and LB^r do not improve during 5 consecutive subgradient iterations, the value of π^r is halved. Note that in Step 1, the initial lower bound is set to zero since the cost coefficients of the problem are non-negative. In Step 6, the subgradient optimization procedure is terminated if the relative error (i.e., $RE^r = (UB^r - LB^r) / LB^r$) becomes smaller than ε(ε=0.001). If $LB^r = UB^r$, then the value of the maximum lower bound coincides with the value of the best feasible solution which must be optimal. Before the subgradient algorithm was run, LRP was reformulated so that all constraints which are to be relaxed are scaled. After the scaling, the right-hand values of these constraints are set to one. This is convenient because relaxing constraints with varying right-hand sides can lead to a constraint with a very large right-hand side dominating the step size expression for T^r in Step 8. Note that if $w^r_{tk} = 0$ and $G^r_{tk} < 0$, then w^r_{tk} will remain zero after the update, but a $G^{r\,2}_{tk}$ factor will be included in the equation for T^r. There seems to be little effective reason for this; it is known that w^r_{tk} is not going to alter. Hence, in Step 7, the subgradients are adjusted before calculating T^r using $G^r_{tk} = 0$, if $w^r_{tk} = 0$ and $G^r_{tk} < 0$. Unless terminated by Step 6 above, the subgradient optimization procedure is terminated when π^r is sufficiently small($\pi^r \leq 0.0005$).

5. Implementation Issues

In addition to the implementation issues discussed in Section 4, several strategies have been employed to enhance the performance of the proposed solution procedure. Let t_l and t_k be the minimum and the maximum job completion times in a schedule. These values are not known in advance. Therefore, to implement the Lagrangian procedure, sufficiently small and large values for t_l and t_k, respectively, should be used to encompass the real values. The earliest time that jobs can complete is the smallest job ready time plus one, and this time is selected as t_l. The maximum job completion time of a feasible schedule that finishes quite late can be taken as t_k. It should be secured that the maximum job completion time of an optimal schedule for a given problem is less than t_k. One way to determine t_k for a given problem is to get an optimal schedule for the case where (i) d_j= d, where $d = \max_{j\in J}\{d_j\}$, j=1,...,n, (ii) $R_k(j) = R_k$, where $R_k = \max_{i\in I}\{R_i(i)\}$, j=1,...,n; k=1,...,u, and (iii) r_j= r, where $r = \max_{i\in I}\{r_j\}$, j=1,...,n. Therefore, an optimal schedule for this simple case can be constructed as follows. s jobs are assigned to machines at moment d, where $s = \min\{\min_{k=1,\ldots,u} \lfloor b_k/R_k \rfloor, m\}$. In sequence, the assignment of jobs to machines is continued at d+1, d-1, d+2,..., until all jobs are assigned. If ready time r becomes a constraint, then jobs are assigned only to the tardy job set. The resulting schedule is optimal for this case. Thus, the maximum completion time of the schedule is $d + \lceil n/(2s) \rceil + \max\{0, r + 1 - [d - \lceil n / (2\times s) \rceil]\}$. This schedule is optimal since, at each moment, it assures the maximal utilization of machines and additional resources, hence minimizing the E-T penalties. Detailed expressions for t_l and t_k can be given as follows:

$$t_l = r + 1,$$
$$t_k = d + \lceil n / (2\times s) \rceil + \max\{0, r + 1 - [d - n \lceil / (2\times s) \rceil]\}.$$

Preliminary computational results were attained by using NETFLO [15] to solve the sub-problem with the time horizon obtained above. This time horizon was too broad and resulted in a very large number of binary variables. Thus, in the experiments performed in the next section, the completion time of the schedule generated by H1 was selected as t_k, and t_l was set to one because at least one ready time was zero and the due dates were small. The use of this heuristic t_k significantly reduced the number of variables. However, if in the solution of a problem at least one job completes at t_k, this problem is solved again after increasing t_k by 3 units to check if the maximum job completion time of the resulting solution coincides with the new t_k. This process is repeated until the maximum job completion time of the resulting solution does not coincide with the new t_k, and this final solution is accepted as the best one. In the preliminary tests, when the t_k obtained from H1 was used, the CPU time was reduced by approximately 60%.

A second adjustment yielded additional CPU time savings. The original NETFLO routine was initially used to solve the transportation sub-problem starting from scratch in each iteration. In this case, it was observed that NETFLO took an average of 84% of the total CPU time to solve the sub-problems. Thus, the original NETFLO was modified to allow starting from the previous optimal solution instead of starting from scratch in solving the updated LRP in each iteration. The use of this modified NETFLO routine reduced the total CPU time by approximately 55% and now the proportion of NETFLO in solving the sub-problem is down to an average of 64%.

A computational study to test the performance of the proposed Lagrangian relaxation algorithm for randomly generated large size problems is presented in the next section. The accuracy of the algorithm is measured by the mean relative percent deviation = $100 \times (UB^* - Z_D(w^*)) / Z_D(w^*)$.

6. Computational Results

The subgradient algorithm shown in the previous section was coded in FORTRAN 77 and run on a MICRON/P5/166 with 16MB. The modified NETFLO routine was used to solve the transportation problem. The subgradient optimization code was built around the NETFLO routine and computational experiments of varying problem sizes were undertaken. The parameters shown in Tables 1-2 were used to select job due dates, job ready times, resource requests for jobs, and the resource limits of u different types of additional resources where u = 1, 2, 3. For example, the job due dates for a test problem with 100 jobs follow a discrete uniform distribution defined by the interval (20, 30). For all test problems, the amounts of resource requests from jobs were generated from a discrete uniform distribution with interval (0,5). The ready time for each job was generated from a uniform distribution with interval (0,10). The amounts of available resources vary based on the number of machines. For each combination of parameters, 10 replications were executed.

〈Table 1〉 Distribution of job parameters for the sample data sets

# of Jobs	Due Dates	Ready Times	Resource Requests
100	(20,30)	(0,10)	(0,5)
200	(30,60)	(0,10)	(0,5)
300	(30,80)	(0,10)	(0,5)

〈Table 2〉 Distribution of resource limits for the sample data sets

# of Machines	# of resource Types	Resource Limits
2	1, 2, 3	(6,10)
3	1, 2, 3	(9,15)
5	1, 2, 3	(15,25)

To evaluate the effectiveness of the proposed algorithm, the mean relative percent deviation, the number of optimal solutions attained, and CPU time (sec.) have been collected. It is observed that the maximum job completion time of the best solution to LRP coincides with the t_k value obtained from H1 in only 5.5% of the test problems. In all these cases, only one time increase of t_k by 3 units is sufficient to check for optimality.

The computational results for all test problems are given in Tables 3. As the results show, the subgradient algorithm performs well based on the mean relative percent deviation. The numbers in parenthesis in the sixth column represent the number of times the algorithm gets a zero gap (i.e., UB=LB). A gap of zero indicates that the algorithm has converged to an optimal solution. The algorithm attains optimal solutions to 174 problems out of 270 tested problems. The algorithm performs exceptionally well for problems with only one type of additional resource where the average % deviation from optimality is less than 0.3%. The average % deviation for all problems is 0.63%.

〈Table 3〉 Computational Results

# of Jobs	Due Dates	Resource Request / Ready Times	# of M/Cs	Resource Types	Avg. % (UB-LB)/LB	Avg. # of Iteration	Avg. CPU Time(sec.)
100	(20,30)	(0,5) / (0,10)	2	1	0.57(8)	10.8	0.79
				2	0.31(5)	26.4	1.92
				3	1.84(3)	28.1	2.68
			3	1	0.31(9)	9.5	0.62
				2	0.59(7)	22.2	1.86
				3	0.81(8)	13.7	1.28
			5	1	0.11(9)	23.1	1.82
				2	0.15(9)	8.5	0.79
				3	0.08(9)	18.3	1.53
200	(30,60)	(0,5) / (0,10)	2	1	0.19(6)	24.0	4.76
				2	0.56(4)	26.0	5.44
				3	1.48(1)	47.5	13.38
			3	1	0.14(8)	20.6	4.50
				2	0(10)	11.1	2.50
				3	1.49(4)	42.6	12.23
			5	1	0(10)	6.2	1.25
				2	0.11(9)	9.5	2.33
				3	0.95(7)	24.8	7.12
300	(30,80)	(0,5) / (0,10)	2	1	0.05(7)	19.1	7.81
				2	0.71(5)	33.1	16.42
				3	0.89(3)	37.9	23.03
			3	1	0.50(6)	29.4	13.78
				2	0.59(4)	27.5	14.21
				3	1.91(2)	42.7	25.78
			5	1	0(10)	3.4	1.81
				2	1.72(5)	33.8	20.20
				3	1.04(6)	36.5	21.23

7. Problem P2

This problem is a special case of problem (P1) in that there exists one single type of additional resource, and each job requires at most one unit of additional resource. If b is the amount of additional resource (units of resource) available in the system at any point in time and there are m identical parallel machines, it is assumed that b 〈 m. Let J_r be the set of jobs such that $J_r=\{j\in J: R(j)=1\}$, and J_0 be the set of jobs such that $J_0=J-J_r$. Note that at most b jobs requiring one unit of additional resource for their processing can be processed simultaneously.

If each job $j\in J_r$ is assigned only to the first b machines, then it is not necessary to consider the resource constraints since any change of schedules on these b machines does not violate the resource constraints. Therefore, this problem can be reduced to a simple zero-one linear programming problem without resource constraints. Let C_{ijt} be the cost of assigning

job j to the (t-1, t) time slot in machine M_i, i=1,2,...,m. If job j is finished before its due date, the earliness cost is caused by (d_j-t). If job j completes its processing after its due date, the tardiness cost is caused by $(t-d_j)$. Note that any job $i \in J_b$ can be assigned to any machine, but any job $j \in J_r$ can only be assigned to the first b machines. The detailed expression of C_{ijt} is as follows:

$$C_{ijt} = \begin{cases} (d_j - t) & \text{if } i=1,...,b; j=1,...,n; t=r_j+1,...,d_j; \text{or} \\ & \text{if } i = b+1,...,m; j=1,...,n; R(j)=0; t=r_j+1,...,d_j; \\ (t - d_j) & \text{if } i=1,...,b; j=1,...,n; t=d_j+1,...,t_h; \text{or} \\ & \text{if } i = b+1,...,m; j=1,...,n; R(j)=0; t=d_j+1,...,t_h; \\ \infty & \text{otherwise.} \end{cases}$$

Now the problem can be formulated as a zero-one ILP problem, and shown to be equivalent to an asymmetric assignment problem. The corresponding transportation network consists of n senders (representing jobs) and $m(t_h-t_l+1)$ receivers (i,t) (representing machines and completion times). In this formulation, C_{ijt} is the unit cost and x_{ijt} is the decision variable representing the flow on arc (j,(i,t)).

$$\text{Min} \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{t=t_l}^{t_h} C_{ijt}\, x_{ijt},$$

〈Table 4〉 Data set for an example problem

J	1	2	3	4	5
$R_i(j)$	1	0	1	1	0
r_j	0	0	0	0	0
d_j	2	1	2	2	1

〈Table 5〉 C_{ijt} for an example problem

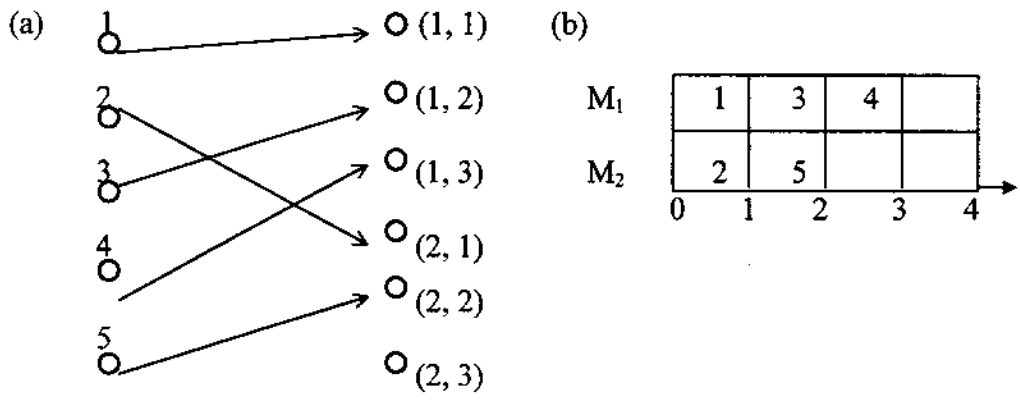
j,(i,t)	(1,1)	(1,2)	(1,3)	(2,1)	(2,2)	(2,3)
1	1	0	1	1	0	1
2	0	1	2	0	1	2
3	1	0	1	1	0	1
4	1	0	1	1	0	1
5	0	1	2	0	1	2

s.t.

$$\sum_{i=1}^{m} \sum_{t=t_l}^{t_h} x_{ijt} = 1, \quad j=1,2,...,n;$$

$$\sum_{j=1}^{n} x_{ijt} \le 1, \quad i=1,2,...,m;\ t=t_l,t_l+1,...,t_h,$$

$$x_{ijt} \ge 0, \quad i=1,2,...,m;\ j=1,2,...,n;\ t=t_l,t_l+1,...,t_h,$$



〈Figure 2〉 (a) Transportation network(only those arcs for which $x_{ijt} = 1$ in the optimal solution are depicted), (b) An optimal schedule.

where $x_{jt} = \begin{cases} 1 & \text{if job j is completed at time t on machine } P_i, \\ 0 & \text{otherwise.} \end{cases}$

It follows that an optimal schedule can be obtained by an assignment algorithm (e.g., the auction algorithm of Bertsekas [16], and the hungarian algorithm of Kuhn [17]). An example of this problem with five independent jobs and two identical parallel machines is provided. In Table 4, job due dates and ready times along with resource requirements are given. The maximum amount of available units of additional resource is one (i.e., b = 1). The cost of job j assigned to machine P_i at time t is also provided in Table 5. A transportation network and an optimal solution of this example are provided in Figure 1.

8. Conclusions

In this work, two different parallel E-T scheduling problems with additional resource have been studied. The first problem is the case in which the number of different types of additional resources is arbitrary, and each job requires an arbitrary number of units of additional resource for each additional resource type at any point in time. In addition, job ready times and job due dates are distinct. All processing times are the same, i.e., p_i= 1,i=1,2,...,n. The model is formulated as a zero-one ILP. A Lagrangian relaxation procedure that uses a subgradient optimization technique has been implemented for this model to obtain near optimal solutions and deviations from optimality for large sized problems. The problem can be generalized by relaxing the assumption that all processing times are the same. To formulate the general problem, constraints which ensure that each job is continuously processed to its completion are needed (i.e., $\sum x_{jk} = p_j$, $p_j y_{jt} = \sum_{k=t-p_j+1}^{t} x_{jk}$, and $\sum y_{jt} = 1$, where x and $y \in \{0,1\}$). These constraints make the problem more difficult. Thus, several interesting cases for future research remain. The second problem is a special case of the first one in that one additional resource type and each job requires at most one unit of additional resource. Based on the observation, the resource constraints are removed and this problem can be reduced to an assignment problem. These models can be easily extended to consider a quadratic objective function where the objective is to minimize the sum of the squared deviations of job completion times with respect to due dates.

[References]

[1] Baker, K.R., Introduction to Sequencing and Scheduling, John Wiley, N.Y., 1974

[2] Uzsoy, R., Lee, C.Y. and Martin-Vega, L.A., "A Review of Production Planning and Scheduling Models in the Semiconductor Industry Part I: System Characteristics, Performance Evaluation and Production Planning," IIE Transactions on Scheduling and Logistics, Vol. 24, pp. 47-61, 1992

[3] Chandru, V., Lee, C.Y. and Uzsoy, R., "Minimizing Total Completion Time on Batch Processing Machines," International Journal of Production Research, Vol. 31, pp. 2097-2121, 1993

[4] Blazewicz, J., "Selected Topics in Scheduling Theory," Annals of Discrete Mathematics, Vol. 31, pp. 1-60, 1987

[5] Garey, M.R. and Johnson, D.S., "Complexity Results for Multiprocessor Scheduling under Resource Constraints," SIAM Journal on Computing, Vol. 4, pp. 397-411, 1975

[6] Blazewicz, J., Barcelo, J., Kubiak, W. and Rock, H., "Scheduling Tasks on Two Processors with Deadlines and Additional Resources," European Journal of Operational Research, Vol. 26, pp. 364-370, 1986

[7] Blazewicz, J., Kubiak, W., Rock, H. and Szwarcfiter, J., "Minimizing Mean Flow-Time with Parallel Processors and Resource Constraints," Acta Informatica, Vol. 24, pp. 513-524, 1987

[8] Baker, K.R. and Scudder, G.D., "Sequencing with Earliness and Tardiness Penalties: A Review," Operations Research, Vol. 38, pp. 22-36, 1990

[9] Blazewicz, J., Cellary, W., Slowinski, R. and Weglarz, J., "Scheduling under Resource Constraints - Deterministic Models in: Peter L. Hammer (eds.)," Annals of Operations Research, Vol. 7, J.C. Baltzer AG, Scientific Publishing Company, Switzerland, 1986

[10] Geofrion, A. M., "Lagrangian relaxation in integer programming", Mathematical Programming Study, Vol. 2, pp. 82-114, 1974

[11] Held, M. and Karp, R.M., "The Traveling-Salesman Problem and Minimum Spanning Tree", Operations Research, Vol. 18, pp. 1138 1162, 1970

[12] Held, M. and Karp, R.M., "The Traveling-Salesman Problem and Minimum Spanning Tree Part II," Mathematical programming, Vol. 1, pp. 6 25, 1971

[13] Held, M., Wolfe, P., and Crowder, H.D., "Validation of Subgradient Optimization," Mathematical programming, Vol. 6, pp. 62 88, 1974

[14] Goffin, J.L., "On the Convergence Rates of Subgradient Optimization Method", Mathematical Programming, Vol. 13, 329 347, 1977

[15] Kennington, J. L. and Helgason, R. V., Algorithms for Network Programming, John Wiley, New York, NY., 1980

[16] Bertsekas, D.P., Linear Network Optimization, Algorithms and Codes, MIT Press, Cambridge, Mass., 1991

[17] Kuhn, N.W., "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, Vol. 2, pp. 83-97, 1955

김대철

1985년 고려대학교 산업공학 학사

1987년 고려대학교 산업공학 석사

1993년 University of Pittsburgh 산업공학 석사

1997년 Pennsylvania State University 산업공학 박사

1997년 Pennsylvania State University 산업공학 Post Doctor

현 재 (주)LG-EDS시스템 컨설팅부문 Supply Chain Solution팀, 차장

관심분야 Production Scheduling, Network 이론, Supply Chain Management

97년 12월 최초접수, 98년 6월 최종수정