The Discrete Resource Allocation Problem in Flow Lines

Selcuk Karabati • Panagiotis Kouvelis • Gang Yu
Bilkent University, Management Department, Faculty of Business Administration, Bilkent, Ankara 06533, Turkey
Duke University, The Fuqua School of Business, Durham, North Carolina 27706
The University of Texas at Austin, Management Science and Information Systems Department, College of Business Administration CBA 5.202, Austin, Texas 78712

In this paper we address the discrete resource allocation problem in a deterministic flow line. We assume that the processing times are convex and nonincreasing in the amount of resources allocated to the machines. We consider the resource allocation problem for a fixed sequence of jobs for various performance criteria (makespan, weighted sum of completion times, cycle time for cyclic schedules), and develop a formulation of the problem as a convex program, where the number of constraints grows exponentially with the number of jobs and machines. We also present a generalization of the formulation for resource allocation problems in acyclic directed graphs. We demonstrate that the problem is NP-complete in the strong sense and present an effective solution procedure. The solution procedure is an implicit enumeration scheme where a surrogate relaxation of the formulation is used to generate upper and lower bounds on the optimal objective function value. Finally, we address the simultaneous scheduling and resource allocation problem, and we present an approximate and iterative solution procedure for the problem.

(*Resource Allocation; Flow Shop Scheduling; Integer Programming; Surrogate Relaxation*)

1. Introduction

The discrete resource allocation problem considered in this paper consists of allocating limited resources among machines of a deterministic flow line with finite or infinite capacity buffers. The objective of the problem is to find an allocation of the resources which optimizes the efficiency of the flow line with respect to certain performance criteria (makespan, weighted sum of completion times, cycle time for cyclic schedules). The modeling framework and the solution procedure for this problem can be generalized to handle a more general class of static resource allocation problems in acyclic directed graphs.

The problem considered in this paper is along the lines of the recent research efforts to broaden the scope of the production scheduling function by capturing some inherent flexibilities of the resources employed in manufacturing. For example, in a flow line with cross-trained workers, the worker allocation problem can be simultaneously solved with the job scheduling problem. Job processing times are usually a function of the amount and mix of resources dedicated to machines, and therefore considering the resource flexibility in making the scheduling decisions can considerably improve the performance of the flow line.

The control of processing times through resource allocation has been studied in the project management context by Moder et al. (1970) and Wiest and Levy (1977). Recently, Daniels and Mazzola (1994) have studied the problem of coordinating scheduling activities and resource allocation tasks in flow shops. They formulate the flexible resource scheduling problem with the objective of simultaneously determining the job sequence, resource allocation policy, and operation start times that optimize the system performance. However, as their approach attempts to optimize over all dynamic allocation policies that consider potential reallocation

0025-1909/95/4109/1417$01.25

of resources at the completion of any operation, the resulting combinatorial problem is extremely difficult. Daniels and Mazzola (1994) present optimal resource allocation results for five-job, four-machine problems. An important aspect of their computational results is that they demonstrate the performance improvements associated with flexible resource scheduling can be substantial in flow shop environments.

The continuous version of the resource allocation problem considered in this paper has been studied in the literature. Luss and Gupta's (1975) ranking algorithm for minimizing the sum of convex and continuously differentiable functions subject to a single budget constraint was later extended by Bitran and Hax (1981) to handle differentiable, but not necessarily strictly convex functions. (For a detailed discussion of the continuous resource allocation problems we refer the interested reader to Ibaraki and Katoh 1988.) Unfortunately, the discrete nature of the resources allocated in the context of our application forces us to study a significantly more difficult combinatorial problem, as our further discussion in this paper will demonstrate.

For high volume flow lines, in which the product mix remains stable over a planning horizon of a few months, static resource allocation policies, which optimize the performance of the flow line for an appropriately selected schedule, are extremely attractive to operations managers. Static resource allocation policies are easy to implement, and enhance the stability of the production environment by minimizing instances of potential interruptions. In this paper we study the static resource allocation problem for flow lines with deterministic processing times.

We consider a flow line with finite or infinite capacity buffers between the machines. The processing times of the various operations are assumed to be convex and nonincreasing in the amount of resources allocated to the machines that perform them. Although the static resource allocation problem has considerably more structure than its dynamic equivalent, it still remains a hard combinatorial problem. We formally proved that the static resource allocation problem for a fixed sequence of jobs is NP-complete in the strong sense for all the performance criteria that we considered. We were, however, able to exploit the special structure of the integer programming formulation of the problem, and solve problems with realistic size in reasonable computational time. Motivated by the efficiency of our algorithmic procedure for the fixed sequence problem, we proceeded to address the simultaneous scheduling and resource allocation problem for flow lines. This problem, because of its difficulty, has never been addressed in the research literature before. We suggest an approximate and iterative solution procedure that can adequately address the problem for realistic size problems.

This paper is organized as follows. In §2 we present a brief overview of the permutation scheduling problem in flow lines, which serves as a necessary background for our further discussion. A formulation of the discrete resource allocation problem in a deterministic flow line and for a fixed sequence of jobs is presented in §3. In §3 we also discuss the computational complexity of the problem and present a generalization of the formulation for resource allocation problems in acyclic directed graphs. In §4 we present a solution procedure for the problem, which is an implicit enumeration scheme that uses a surrogate relaxation of our formulation to generate tight lower and upper bounds. In §5, through extensive numerical experimentations, we demonstrate the effectiveness of the proposed solution procedure. In §6 we propose an approximate and iterative solution procedure for the simultaneous resource allocation and cyclic scheduling problem. Finally, in §7 we present our concluding remarks.

2. Directed Graph Representation of Completion Times and Other Performance Measures in Flow Lines

In this section we present a brief overview of the permutation scheduling problem in a deterministic flow line. This overview provides to the reader the necessary background for understanding our methodology for the discrete resource allocation problem in flow lines for a fixed sequence of jobs.

The flow line consists of m machines in series, M_1, $M_2, \ldots, M_m$, with infinite capacity buffers between the machines. Without any loss of generality, we assume that n jobs, $J_1, J_2, \ldots, J_n$, are scheduled according to

the permutation sequence $\{1, 2, \ldots, n\}$. We denote by $p_{i,j}$, $i = 1, \ldots, n; j = 1, \ldots, m$, the processing time of job J_i on machine M_j. We can now represent this schedule by the directed graph $G(V, A)$ (see Figure 1), where

$$V = \{(i, j), i = 1, 2, \ldots, n; j = 1, 2, \ldots, m\}, \quad \text{and}$$

$$A = \{((i, j), (i + 1, j)), ((i, j), (i, j + 1)), \\ i = 1, 2, \ldots, n; j = 1, 2, \ldots, m\}.$$

We note that in $G(V, A)$ vertex (i, j) corresponds to the operation of job J_i on machine M_j (for an original reference to this framework see Monma and Rinnooy Kan 1983). We also define the weight $w_{(i,j)}$ of vertex (i, j) as the processing time of job J_i on machine M_j, i.e., $w_{(i,j)} = p_{i,j}$. Given $G(V, A)$ defined as above, the completion time $C_{i,j}$ of job J_i on machine M_j is equal to the weight of the maximum-weighted directed path from vertex (1, 1) to vertex (i, j) in $G(V, A)$. Let $T_{i,j}$ be the set of paths from vertex (1, 1) to vertex (i, j) in $G(V, A)$. Note that each path in $T_{i,j}$ consists of $(i + j - 1)$ ordered vertices from vertex (1, 1) to vertex (i, j). Let

$$\tau_{i,j} = (v_1^{\tau_{i,j}}, v_2^{\tau_{i,j}}, \ldots, v_{i+j-1}^{\tau_{i,j}})$$

be a path in $T_{i,j}$. The weight $W(\tau_{i,j})$ of path $\tau_{i,j}$ is given by

$$W(\tau_{i,j}) = \sum_{k=1}^{i+j-1} w_{v_k^{\tau_{i,j}}}. \tag{1}$$

Using the above representation of path lengths, $C_{i,j}$ can be written as

$$C_{i,j} = \max_{\tau_{i,j} \in T_{i,j}} W(\tau_{i,j}). \tag{2}$$

A path $\tau_{i,j} \in T_{i,j}$ that has its weight equal to $C_{i,j}$ is referred to as a critical path for vertex (i, j).

We can now define the performance measures of makespan (MS) and weighted sum of completion times (MC) as follows:

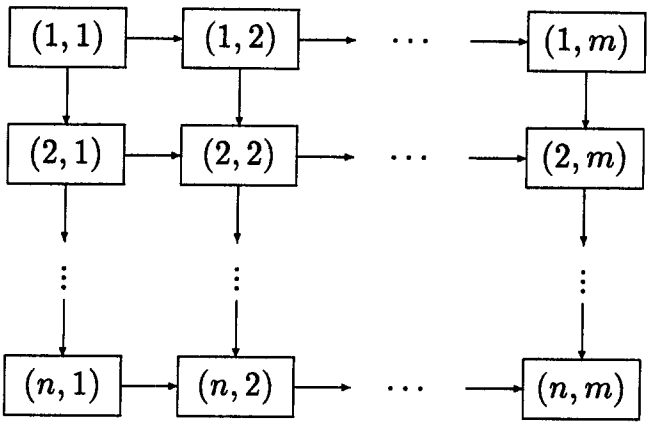
$$MS = \max_{\tau_{n,m} \in T_{n,m}} W(\tau_{n,m}), \quad \text{and} \tag{3}$$

$$MC = \sum_{i=1}^{n} r_i \left(\max_{\tau_{i,m} \in T_{i,m}} W(\tau_{i,m}) \right), \tag{4}$$

where r_i is the weight of job J_i, $i = 1, 2, \ldots, n$.

The above graph theoretic framework can be extended to include finite capacity buffers (see Appendix A). For simplicity of presentation we restrict our attention to the infinite capacity buffers case. However, the solution methodology and the complexity results of this paper can be readily generalized to the finite capacity buffers case. A further extension of the framework is for the cycle time performance criterion in cyclic scheduling environments. In the cyclic scheduling problem, a set of jobs (referred to as the Minimal Part Set) is produced in a repetitive manner, and the objective is to minimize the cycle time, which is equivalent to maximizing the throughput rate of the flow line in the steady state (McCormick et al. 1990 and Karabati and Kouvelis 1991). In this case the directed graph $G(V, A)$ of Figure 1 also includes arcs from the vertex (n, j) to the vertex $(1, j)$ for all j. Finding the cycle time is equivalent to finding the maximum-weighted path around the cylinder formed by the modified $G(V, A)$. For details on the modeling framework for cyclic schedules the interested reader is referred to Karabati and Kouvelis 1991.

Figure 1 **Directed Graph $G(V, A)$ Representation of Completion Times for the Infinite Capacity Buffers Case**



3. Formulation of the Discrete Resource Allocation Problem for a Fixed Sequence of Jobs

In this section we present a formulation of the discrete resource allocation problem for a fixed sequence of jobs.

We will use the term *worker* to refer to the resource we want to allocate without any loss of generality. Let N be the total number of workers to be allocated, x_j be

the number of workers allocated to machine M_j, $j = 1, 2, \ldots, m$, and $X = (x_1, x_2, \ldots, x_m)$ be an allocation vector. We assume that workers are identical, i.e., a worker can be assigned to any one of the machines in the flow line. We now let $p_{i,j}(x_j)$ be the processing time of job J_i, $i = 1, 2, \ldots, n$, on machine M_j, $j = 1, 2, \ldots, m$, when x_j workers are allocated to machine M_j. We assume that $p_{i,j}(\cdot)$ is convex and nonincreasing in x_j, reflecting the diminishing effect of additional workers. We also assume that $0 \le p_{i,j}(0) < \infty$, $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$, therefore x_j denotes the number of *additional* workers allocated to machine M_j, $j = 1, 2, \ldots, m$. We can now formulate the problem as a general resource allocation problem as follows:

$$\text{(DRAP)}: \quad f_{\text{DRAP}} = \min F(X)$$

$$\text{subject to} \quad \sum_{j=1}^{m} x_j \le N,$$

$$x_j \in \{0, 1, 2, \ldots, N\}, \quad j = 1, 2, \ldots, m,$$

where $F(X)$ denotes the value of the performance measure when we allocate workers using allocation vector $X = (x_1, x_2, \ldots, x_m)$. Without any loss of generality in the presentation of our solution methodology, and for the convenience of the reader, we will assume that the objective is to minimize the makespan of the fixed schedule.

As we have discussed in §2, the makespan of the fixed schedule is equal to the weight of the maximum-weighted path in the path set $T_{n,m}$. (For brevity, we let $T \equiv T_{n,m}$, $\tau \equiv \tau_{n,m}$, and $v_k^\tau \equiv v_k^{\tau_{n,m}}$.) For a given allocation vector X, the weight of path τ, $\tau \in T$ is denoted by $f(\tau, X)$ and given by the following relationship:

$$f(\tau, X) = \sum_{k=1}^{n+m-1} w_{v_k^\tau}(X), \tag{5}$$

where for a vertex $v_k^\tau = (i, j)$, $w_{v_k^\tau}(X)$ denotes the weight of vertex (i, j) under allocation X, and is given by $w_{v_k^\tau}(X) = w_{(i,j)}(X) = p_{i,j}(x_j)$. Therefore $f(\tau, X)$ can be rewritten as

$$f(\tau, X) = \sum_{i=1}^{n} \sum_{j=1}^{m} p_{i,j}(x_j) 1\{(i, j) \in \tau\}, \tag{6}$$

where $1\{(i, j) \in \tau\}$ is an indicator function, and is equal to 1 if vertex (i, j) is in path τ, 0 otherwise. By regrouping the terms in (6) according to the machine indices we obtain

$$f(\tau, X) = f_1^\tau(x_1) + f_2^\tau(x_2) + \cdots + f_m^\tau(x_m), \tag{7}$$

where $f_j^\tau(x_j)$ denotes the contribution of machine M_j to the weight of path τ, $\tau \in T$, when x_j workers are allocated to this machine. The term $f_j^\tau(x_j)$, $j = 1, 2, \ldots, m$, can be written as follows:

$$f_j^\tau(x_j) = \sum_{i=1}^{n} p_{i,j}(x_j) 1\{(i, j) \in \tau\}. \tag{8}$$

Note that $f_j^\tau(x_j)$ is the sum of individual processing times, which are convex and nonincreasing in the number of workers allocated to machine M_j, therefore $f_j^\tau(x_j)$ is also convex and nonincreasing in x_j. Since the makespan of the problem is equal to the weight of the maximum-weighted path in path set T, (DRAP) can be rewritten as follows:

$$\text{(DRAP)}: \quad f_{\text{DRAP}} = \min M$$

$$\text{subject to} \quad \sum_{j=1}^{m} f_j^\tau(x_j) \le M, \quad \tau \in T,$$

$$\sum_{j=1}^{m} x_j \le N,$$

$$x_j \in \{0, 1, 2, \ldots, N\}, \quad j = 1, 2, \ldots, m.$$

We note that in an optimal solution to (DRAP), the resource availability constraint holds as an equality. This follows directly from the fact that the processing times are convex and nonincreasing in the number of workers allocated to the machines, and therefore, there is a potential reduction, and definitely no increase, of the makespan by the allocation of additional workers. When the path set T contains a single path, i.e., $|T| = 1$, (DRAP) becomes a minisum resource allocation problem with convex, separable, and nonincreasing cost functions, and it can be solved using a simple greedy algorithm (for a detailed discussion of the minisum resource allocation problem see Ibaraki and Katoh 1988). However, when the size of the path set $|T|$ is greater than one, which is the case for the problem considered in this paper, (DRAP) is NP-complete in the strong sense for any of the performance criteria we considered as the following theorem demonstrates.

THEOREM 1. *(DRAP) is NP-complete in the strong sense, even when $f_j^\tau(\cdot)$ is linear.*

We present the proof of Theorem 1 in Appendix B.

The discrete resource allocation problem in flow lines addressed in this paper can be considered as a special case of a more general discrete resource allocation problem in acyclic directed graphs. The statement of this general problem is as follows. Let $G(V, A)$ be an acyclic directed graph, and v_s and v_e be the start and end vertices in the linear ordering of the vertices of the acyclic graph. Such a linear ordering of the vertices of an acyclic graph is always possible (see Lawler 1976). Let $G_k(V_k, A_k)$, $k = 1, 2, \ldots, K$, be a subgraph of $G(V, A)$, such that V_k is a set of ordered vertices, and

$$V_k \subset V, \quad k = 1, 2, \ldots, K; \quad \bigcup_{k=1}^{K} V_k = V;$$

$$V_k \bigcap V_l = \varnothing, \quad k, l = 1, 2, \ldots, K; \quad k \neq l,$$

and $A_k = \{(v_j, v_l) \in A \mid v_j, v_l \in V_k\}$. Note that $G_k(V_k, A_k)$ is a path between two vertices of V. Weights $w_l(x_k)$'s are associated with each vertex $v_l \in V_k$ and are assumed to be nonincreasing convex functions of the resource units x_k allocated to V_k. We are interested in allocating N resource units to the subsets V_k, $k = 1, 2, \ldots, K$, so that a performance measure $F(X)$ representing the weight of the maximum-weighted path from v_s to v_e is minimized.

Observe that the discrete resource allocation problem in flow lines defines V_k as the operations performed on machine M_k (i.e., $V_k = ((i, k), i = 1, 2, \ldots, n)$ according to our flow line notation). Another interesting special case of the resource allocation problem in an acyclic graph is obtained by defining the subset V_k as the operations associated with a specific job J_k (i.e., $V_k = ((k, j), j = 1, 2, \ldots, m)$), and thus the allocation of the resources is done now in a static way to the various jobs rather than the machines. Finally, another interesting special case is to treat each vertex of the original graph as a subset, and thus, in our flow line terminology, decide on how to allocate resources to individual operations. This last special case models also the nonrenewable resource allocation problem to the various job tasks in a project scheduling environment (for further motivation of the application see Wiest and Levy 1977).

In our further discussion, we present the solution methodology for the discrete resource allocation problem in flow lines. However, the same methodology generalizes in a straight forward manner for the general discrete resource allocation problem in acyclic directed graphs.

4. A Solution Procedure for the Fixed Sequence Problem

In this section we present solution procedures for (DRAP). Sections 4.1–4.3 deal with different aspects of an implicit enumeration approach, where a surrogate relaxation of (DRAP) is used to develop lower and upper bounds on the optimal value of the objective function. Section 4.4 discusses the use of a greedy heuristic procedure for the solution of the problem.

4.1. The Surrogate Relaxation Approach

We construct the surrogate relaxation of (DRAP) as follows. Let $\mu_\tau \geq 0$ be a multiplier for the constraint that corresponds to path τ, $\tau \in T$. Let $\mathcal{M} = \{\mu \in \mathcal{R}^{|T|} \mid \sum_{\tau \in T} \mu_\tau = 1; \mu_\tau \geq 0, \tau \in T\}$ be a set of multiplier vectors. Using multiplier vector μ, $\mu \in \mathcal{M}$, we obtain the following relaxation of (DRAP):

$$\text{(DRAS)}: \quad f_{\text{DRAS}}(\mu) = \min M$$

$$\text{subject to} \quad \sum_{j=1}^{m} \left(\sum_{\tau \in T} \mu_\tau f_j^\tau(x_j) \right) \leq M,$$

$$\sum_{j=1}^{m} x_j \leq N,$$

$$x_j \in \{0, 1, 2, \ldots, N\}, \quad j = 1, 2, \ldots, m.$$

Note that (DRAS) is a minisum resource allocation problem with discrete decision variables, and, since the left-hand side of the first constraint is a convex combination of convex and nonincreasing functions (or stated otherwise, if we replace M in the objective function with the left-hand side of the first constraint, the resulting objective function is convex and nonincreasing), it can be solved with the use of a simple greedy procedure (Ibaraki and Katoh 1988). Let μ^* be a multiplier vector such that

$$f_{\text{DRAS}}(\mu^*) = \max_{\mu \in \mathcal{M}} f_{\text{DRAS}}(\mu).$$

$f_{\text{DRAS}}(\mu^*)$ is the value of the best lower bound that can

be obtained using this surrogate relaxation approach, i.e., for all μ, $\mu \in \mathcal{M}$, we have $f_{DRAS}(\mu) \le f_{DRAS}(\mu^*) \le f_{DRAP}$.

4.2. Determination of the Surrogate Multipliers

We now present a heuristic procedure to determine the multiplier vector μ^* for (DRAS). We first consider the following problem:

$$\text{(DRAS')}: \quad f_{DRAS'}(\lambda) = \min M$$

$$\text{subject to} \quad \sum_{j=1}^{m} (\lambda f_j^1(x_j) + (1-\lambda) f_j^2(x_j)) \le M,$$

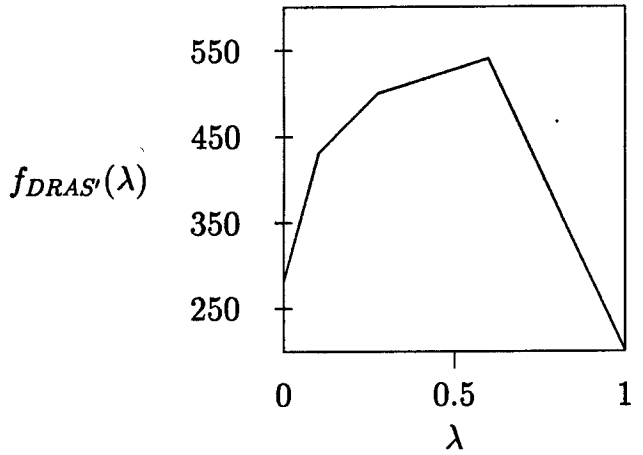
$$\sum_{j=1}^{m} x_j \le N,$$

$$x_j \in \{0, 1, 2, \ldots, N\}, \quad j = 1, 2, \ldots, m,$$

where λ is a scalar such that $0 \le \lambda \le 1$. The first constraint of (DRAS') is a convex combination of two constraints of (DRAP). Note that these two constraints themselves can be surrogations of constraints associated with different paths in path set T of (DRAP). (DRAS') is equivalent to a minisum resource allocation problem which can be solved with the use of a simple greedy procedure. It is a simple task to show that $f_{DRAS'}(\lambda)$ is piecewise linear and concave in λ. In Figure 2 we present the form of $f_{DRAS'}(\lambda)$ for two arbitrarily chosen constraints of a 10-job, 5-machine, 15-worker problem. Let λ^* be a scalar such that

$$f_{DRAS'}(\lambda^*) = \max_{0 \le \lambda \le 1} f_{DRAS'}(\lambda).$$

Figure 2 **Functional Properties of $f_{DRAS}(\lambda)$**



Since $f_{DRAS'}(\lambda)$ is concave in λ, λ^*, or an ϵ-neighborhood of λ^*, can be determined using a simple line search approach on the (0, 1) interval (see Bazaraa and Shetty 1979 for a discussion of the line search procedures).

This line search procedure can be used in a heuristic procedure for computing the multiplier vector μ^* for (DRAS) as follows: Let X_{λ^*} be the worker allocation associated with $f_{DRAS'}(\lambda^*)$. Let τ be the critical path in the path set T under allocation X_{λ^*}. We then let $f_j^1(x_j) = \lambda^* f_j^1(x_j) + (1-\lambda^*) f_j^2(x_j)$ and $f_j^2(x_j) = f_j^\tau(x_j)$, $j = 1, 2, \ldots, m$, and solve (DRAS') with this new surrogate constraint. We proceed the same way until either no improvement in the value of $f_{DRAS'}(\lambda^*)$ is obtained, or (in order to reduce the computational effort to obtain a lower bound) the improvement falls below a prespecified level $\delta > 0$. This procedure is not guaranteed to converge to μ^*; however, as we will demonstrate in §5, the lower bounds obtained using the multiplier vectors generated by this procedure are very tight. We can formally state the surrogation procedure as follows:

Procedure Surrogate:

Step 0. Let τ^1 and τ^2 be initially chosen two paths in path set T. Let $f_j^1(x_j) = f_j^{\tau^1}(x_j)$, $f_j^2(x_j) = f_j^{\tau^2}(x_j)$, $j = 1, 2, \ldots, m$. Also let $k = 1$, $f_{DRAS'}^0(\lambda_0^*) = -\infty$, and go to Step 1.

Step 1. Determine $f_{DRAS'}^k(\lambda_k^*)$ using a line search procedure. Let $X_{\lambda_k^*}$ be the resulting allocation. If $f_{DRAS'}^k(\lambda_k^*) - f_{DRAS'}^{k-1}(\lambda_{k-1}^*) > \delta$ go to Step 2, otherwise stop; $\lambda_k^* f_j^1(x_j) + (1-\lambda_k^*) f_j^2(x_j)$, $j = 1, 2, \ldots, m$, is a surrogate constraint for (DRAS), and $f_{DRAS'}^k(\lambda_k^*)$ is a lower bound on the optimal objective function value of the resource allocation problem.

Step 2. Determine the critical path of the fixed schedule under allocation vector $X_{\lambda_k^*}^*$. Let τ_k be this critical path. Let $f_j^1(x_j) = \lambda_k^* f_j^1(x_j) + (1-\lambda_k^*) f_j^2(x_j)$ and $f_j^2(x_j) = f_j^{\tau_k}(x_j)$, $j = 1, 2, \ldots, m$. Also let $k = k + 1$, and go to Step 1.

The above procedure to find a surrogate constraint for (DRAS) involves the solution of a number of minisum discrete resource allocation problems, each having a complexity of $O(\max\{m, N\} \log m)$ when a greedy solution procedure is used to solve them. A continuous relaxation based procedure solves the minisum resource allocation problem in $O(\mathcal{T} + m)$ time, where $\mathcal{T}$ denotes the time to solve the continuous relaxation of the prob-

lem (see Ibaraki and Katoh 1988 for a detailed discussion of alternative solution procedures for the minisum resource allocation problem). In Step 1, we solve a number of minisum resource allocation problems in our line search procedure to determine λ_i^*. Note that the number of problems we need to solve in Step 1 is a function of the desired solution quality, i.e., the value of ϵ. After we update the surrogate constraint in Step 2, Step 1 is visited one more time, and we repeat these steps until a good lower bound is obtained. Similarly, the number of times Step 1 is visited is a function of the desired lower bound quality, which is specified by δ, therefore the trade-off between the quality of lower bounds and the computational effort required to compute them can be managed by appropriately selecting parameters ϵ and δ. We report the details of our computational experience with these procedures in §5.

4.3. Branching Strategy

In our branch-and-bound tree, each node is defined by two row vectors. Let $l = (l_1, l_2, \ldots, l_m)$ be a vector of lower bounds on the possible values of $X = (x_1, x_2, \ldots, x_m)$. Similarly, let $u = (u_1, u_2, \ldots, u_m)$ be a vector of upper bounds on the possible values of $X = (x_1, x_2, \ldots, x_m)$. Let $X = (a_1, a_2, \ldots, a_m)$ be a feasible allocation of workers to machines for the node under consideration. We now branch from this node using allocation vector X by creating the following nodes:

Node	Lower and Upper Bounds
1	$l^1 = (l_1, l_2, l_3, \ldots, l_m)$ $u^1 = (a_1 - 1, u_2, u_3, \ldots, u_m)$
2	$l^2 = (a_1 + 1, l_2, l_3, \ldots, l_m)$ $u^2 = (u_1, u_2, u_3, \ldots, u_m)$
3	$l^3 = (a_1, l_2, l_3, \ldots, l_m)$ $u^3 = (a_1, a_2 - 1, u_3, \ldots, u_m)$
4	$l^4 = (a_1, a_2 + 1, l_3, \ldots, l_m)$ $u^4 = (a_1, u_2, u_3, \ldots, u_m)$
5	$l^5 = (a_1, a_2, l_3, \ldots, l_m)$ $u^5 = (a_1, a_2, a_3 - 1, \ldots, u_m)$
6	$l^6 = (a_1, a_2, a_3 + 1, \ldots, l_m)$ $u^6 = (a_1, a_2, u_3, \ldots, u_m)$
⋮	⋮
$2(m-1)-1$	$l^{2(m-1)-1} = (a_1, a_2, a_3, \ldots, a_{m-2}, l_{m-1}, l_m)$ $u^{2(m-1)-1} = (a_1, a_2, a_3, \ldots, a_{m-2}, a_{m-1} - 1, l_m)$
$2(m-1)$	$l^{2(m-1)} = (a_1, a_2, a_3, \ldots, a_{m-2}, a_{m-1} + 1, l_m)$ $u^{2(m-1)} = (a_1, a_2, a_3, \ldots, a_{m-2}, u_{m-1}, u_m)$

Note that, together with allocation $X = (a_1, a_2, \ldots, a_m)$, the above $2(m-1)$ mutually exclusive nodes constitute an exhaustive partition of the original node. Some of these nodes may be infeasible (e.g., $l_j > u_j$, for some j, $j = 1, 2, \ldots, m$, or $\sum_{j=1}^m l_j > N$, or $\sum_{j=1}^m u_j < N$) and can be eliminated from further consideration. Our initial node is defined by vectors $l = (0, 0, \ldots, 0)$ and $u = (N, N, \ldots, N)$, and at each node we use the allocation vector associated with the solution of $f_{\text{DRAS}}(\mu^*)$ in partitioning the node under consideration into new nodes. When there are more than one candidate node for further branching, we choose the node which has the smallest lower bound value. In each node, in order to reduce the computational effort, we start the surrogate relaxation procedure with the surrogate constraint of the node from which the node under consideration is created.

Our solution procedure for the discrete resource allocation problem in flow lines is a branch-and-bound method which uses the above described branching rule and the surrogate relaxation approach of §4.1 in generating lower and upper bounds on the optimal objective value of the problem. Note that in Step 1 of Procedure Surrogate, as we solve the minisum resource allocation problems, we generate feasible solutions for (DRAP), and these solutions can be used to find upper bounds on the optimal objective function value of the problem. For each node of the search tree we can develop a lower bound using the above described surrogate relaxation approach; however, in solving (DRAS) we now consider additional constraints in the following form:

$$l_j \le x_j \le u_j, \quad j = 1, 2, \ldots, m.$$

We note that the minisum resource allocation problem with lower and upper bounds on the decision variables can still be solved using a simple greedy procedure (see Ibaraki and Katoh 1988); therefore, the inclusion of these constraints in (DRAS) does not increase the complexity of our lower bounding procedure.

4.4. The Greedy Procedure

As we have pointed out in §4.2, the discrete resource allocation problem with a separable and convex objective function, and a single resource constraint can be solved using a greedy (or incremental) solution procedure. The greedy procedure allocates the available

resources sequentially by maximizing the decrease of the objective function after each assignment. In more general resource allocation problems, the greedy procedure results in an optimal solution if the constraints determine a polymatroid and the objective function is weakly convex (Federgruen and Groenevelt 1986). Unfortunately, the discrete resource allocation problem discussed in this paper does not satisfy these necessary properties and the greedy procedure may fail to find an optimal solution, as the following example demonstrates.

EXAMPLE 1. Consider a 3-machine, 3-job, and 3-worker problem with zero capacity buffers. The performance criterion is the cycle time. Processing times are given as $p_{i,j}(x_j) = a_{i,j}/(1 + x_j)$, where $a_{1,1} = 9$, $a_{1,2} = 4$, $a_{1,3} = 6$, $a_{2,1} = 1$, $a_{2,2} = 7$, $a_{2,3} = 9$, $a_{3,1} = 6$, $a_{3,2} = 9$, $a_{3,3} = 1$. In the first iteration of the greedy procedure we compare allocations $X = (1, 0, 0)$, $X = (0, 1, 0)$, and $X = (0, 0, 1)$. In this iteration the best solution is provided by allocation $X = (0, 1, 0)$ with a cycle time of 17. In the second iteration we compare allocations $X = (1, 1, 0)$, $X = (0, 2, 0)$, and $X = (0, 1, 1)$. In the second iteration the best allocation is $X = (0, 2, 0)$ with a cycle time of 16.333. In the last iteration of the greedy procedure we compare allocations $X = (1, 2, 0)$, $X = (0, 3, 0)$, and $X = (0, 2, 1)$. The greedy procedure results in allocation $X = (0, 3, 0)$ with a cycle time of 16. The optimal allocation for this problem is equal to $X^* = (1, 1, 1)$ with a cycle time of 10.

Although the greedy procedure may fail to provide optimal solutions, it can be used as a heuristic procedure for solving the problem. We report our experience with the performance of the greedy heuristic in §5.

5. Computational Experiments

In this section we evaluate the quality of the lower bounds, the effectiveness of the proposed branching rules, and the performance of the greedy heuristic.

We have performed the computational experiments on four different problem classes. Each problem class is defined by the form of the processing time function $p_{i,j}(\cdot)$, and the parameters of the function as follows:

Class A. In this problem class, all processing time functions are of the following form:

$$p_{i,j}(x_j) = \frac{a_{i,j}}{1 + x_j}, \quad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, m,$$

where $a_{i,j}$ denotes the processing time of job J_i on machine M_j, when no additional workers are assigned to machine M_j. In this problem class, $a_{i,j}$'s are generated using a uniform (1, 100) distribution.

Class B. In this problem class, the form of the processing time functions is the same as in Problem Class A; however, $a_{i,j}$'s are generated in a slightly different way. Let $\hat{a}_{i,j}$, $i = 1, 2, \ldots, n; j = 1, 2, \ldots, m$, be random numbers drawn from a uniform (1, 100) distribution. Let $A_j = \sum_{i=1}^{n} \hat{a}_{i,j}$ be the total workload of machine M_j, $j = 1, 2, \ldots, m$, with these initial processing times, i.e., before any worker assignments are made. Let $A_{\max} = \max_{1 \le j \le m} A_j$. We now define the parameters of this problem as follows:

$$a_{i,j} = \hat{a}_{i,j} \frac{A_{\max}}{A_j}, \quad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, m.$$

Note that this normalization of the initial processing times generates a balanced problem, i.e., a problem with equal machine workloads.

Class C. In this problem class, processing time functions have the following form:

$$p_{i,j}(x_j) = a_{i,j} \exp(-b_{i,j} x_j), \quad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, m,$$

where $a_{i,j}$'s are generated using a uniform (1, 100) distribution, and $b_{i,j} = 0.5$, $i = 1, 2, \ldots, n; j = 1, 2, \ldots, m$.

Class D. In this problem class, processing time functions have the same form as in Problem Class C, and $a_{i,j}$'s are generated as described in Problem Class B.

In order to evaluate the quality of the lower bounds generated by the proposed procedure, we created 100 problems in each problem class with 15 jobs, 10 machines, and different numbers of workers ($N = 10$, 15, and 20). We report results on zero capacity buffer problems since these problems are computationally more demanding than the infinite capacity buffer problems. Having zero capacity buffers among the machines increases the difficulty of the problem, because this results in a high degree of interaction among the machines. The performance criterion is chosen as the cycle time because, for the zero capacity buffers case, the structure of set T is much more complicated for the cycle time performance criterion due to the increased number of paths in $G(V, A)$ of the problem. In Table 1 we report

the gap between the *upper* and *lower* bound values of the initial node as a fraction of the initial *lower* bound value. For all problem classes we used $\epsilon = 0.002$, $\delta = 0.5$, which resulted in a good balance between the quality of lower bounds and the computational effort required to generate them. The line search method used in Step 1 of Procedure Surrogate was the *Golden Section Method* (see Bazaraa and Shetty 1979, p. 259). For each problem set we report the minimum, average, and maximum values (over 100 problems) of the relative gap. The average value of the relative gap is around 9–14% for all problem classes, indicating the very satisfactory tightness of the initial lower and upper bounds.

We then proceeded with another set of experiments to measure the effectiveness of the branching rules we proposed in the previous section. For each problem class we created 15-job, 10-machine problems with zero capacity buffers among the machines, and with different numbers of workers (N = 10, 15, and 20). In Table 2 we report the average performance of our branch-and-bound procedure on different problem classes. For each problem set we consider 20 problems, and report the average CPU time (on a multi-user IBM 3081-D), the average number of nodes created in the search tree, and the average maximum number of active nodes, i.e., nodes that are being stored for further branching during the search. We note that the maximum number of active nodes is a good indicator of the storage requirements of the branch-and-bound procedure. The results of Table 2 clearly demonstrate the effectiveness of the proposed procedure. For example for m = 10 and N = 20, the number of possible allocations is

Table 1 Evaluation of the Lower Bound Quality

Number of Workers	Relative Gap	Problem Class A	B	C	D
10	Minimum	0 0073	0 0273	0 0546	0 0502
	Average	0 1068	0.1295	0 1276	0 1323
	Maximum	0 3185	0.3433	0.2467	0 2456
15	Minimum	0 0379	0.0490	0.0770	0.0741
	Average	0 1195	0 1073	0.1419	0.1331
	Maximum	0 2139	0.2163	0.2695	0.2225
	Minimum	0 0246	0.0333	0.0603	0 0436
	Average	0 0963	0.1033	0.1338	0.1438
20	Maximum	0.2297	0.1981	0 2800	0 2490

Table 2 Performance of the Branch-and-Bound Procedure

Problem Class	CPU (seconds)	Number of Nodes	Maximum Number of Active Nodes
		Number of Workers = 10	
A	1.60	127.45	7.50
B	1.27	96 70	5.65
C	6 42	252.25	15.65
D	6 30	250 25	13 45
		Number of Workers = 15	
A	7 43	600 80	32.65
B	5 46	434.50	22 30
C	19.35	680.60	38 65
D	17.31	585.00	31.50
		Number of Workers = 20	
A	5.77	388 35	19 40
B	5.81	365 10	18 10
C	15.22	397 45	21 10
D	12 02	285 10	16 35

$$\binom{N+m-1}{m-1} = \binom{29}{9} \approx 5 \times 10^7,$$

and our procedure finds the optimal solution by generating fewer than 1,000 nodes. We also note that the performance of the branch-and-bound procedure is not sensitive to the choice of the parameters we have used in defining the problem classes.

The results of Table 2 indicate that problems with 15 workers are more difficult to solve than the 20-worker problems, although the number of possible allocations is much smaller for 15-worker problems. We note that in Table 2 we consider 10-machine problems, and in most of the problems optimal allocations are such that the workers are fairly evenly distributed across the line to avoid bottlenecks. For 10-machine problems, it is much easier to create a fairly even allocation with 10 or 20 workers than it is with 15 workers, and this is the major reason for the difficulty associated with 15-worker problems.

As we have discussed in §4.4, the greedy procedure can be used as a heuristic for solving (DRAP). In order to evaluate the performance of the greedy procedure, we created 100 problems in each problem class with 15 jobs, 10 machines with zero capacity buffers, and different numbers of workers (N = 10, 15, and 20). In

Table 3 Evaluation of the Performance of the Greedy Heuristic

Number of Workers	Percent Deviation	Problem Class A	B	C	D
10	Minimum	0	0	0	0
	Average	12 14	12 98	7 96	8 65
	Maximum	93 73	47 02	59 71	33 53
15	Minimum	0	0	0	0
	Average	8.02	6.24	6.51	4.27
	Maximum	127 52	62.68	91 00	35.62
20	Minimum	0	0	0	0
	Average	12 30	5 91	9 73	8 49
	Maximum	190.60	19.92	163.31	35 70

Table 3 we report the performance of the greedy procedure as percent deviations from optimal solution values. For each problem set we report the minimum, average, and maximum values of the deviations (over 100 problems). Although the maximum deviations are considerably high, especially in Problem Classes A and C, the average performance of the greedy heuristic is quite satisfactory.

As we have pointed out earlier, our procedure solves the discrete resource allocation problem for a fixed sequence of jobs, and therefore the selection of the initial sequence of jobs is an important factor that may affect the solution quality. In order to analyze this issue, we created a single 15-job, 10-machine, and 15-worker problem for each problem class, and solved each problem for 100 randomly selected sequences. In Table 4 we report the maximum, average, and minimum objective function values over these 100 sequences. The results of Table 4 indicate that, for all problem classes, the optimal value of the objective function is very sensitive to the choice of the initial sequence of jobs. This also points to the high degree of dependency between scheduling and resource allocation decisions, in terms of achieving a high throughput rate, i.e., a low cycle time value. We will further discuss this issue in §6, where we present an iterative procedure for the simultaneous solution of the scheduling and resource allocation problems.

Table 4 Sensitivity Analysis of the Objective Function Value

Problem Class	Objective Function Value (Cycle Time) Minimum	Average	Maximum
A	570 921	588 026	606 255
B	616.646	637.574	649.376
C	458.000	472.270	484.500
D	668.453	705 123	730 983

6. Simultaneous Scheduling of Jobs and Optimal Allocation of Resources Problem

In this section we discuss the simultaneous scheduling and discrete resource allocation problem and present an approximate and iterative solution procedure. The cyclic scheduling problem considered in this study has been shown to be NP-complete for fixed processing times (McCormick et al. 1987). For reasons mentioned in the previous section, we concentrate our discussion to cyclic scheduling problems with the cycle time criterion. However, the methodology and the nature of

Table 5 Performance of Procedure Iterative: 6-Job Problems

Problem Class	Problem No	Number of Iter.	CPU (sec)	Objective Fn Initial	Final	Optimal
A	1	2	1.41	270.80	259 04	244 66
	2	2	1 18	288.18	274 17	271 47
	3	3	1 52	259.65	229 40	229 40
	4	1	0 62	265 44	265 44	256.41
	5	2	1 23	281 80	279.39	271.94
B	1	2	1.26	344.81	324 51	324 51
	2	2	1.40	419 69	397.93	394 22
	3	2	1 08	313 65	312 75	312 75
	4	2	1 12	331 95	322 95	315.21
	5	2	1.14	282 42	279 92	279 92
C	1	2	1.45	263 32	252 66	250.00
	2	2	1.24	275 32	271.00	270 00
	3	2	1.68	247 32	236 16	236.16
	4	1	1 17	237.32	237 32	223.32
	5	2	1 42	273.32	268 66	256 32
D	1	2	1 48	350 02	334 70	334 70
	2	1	0.62	281.44	281 44	268 22
	3	1	0 83	298 54	298 54	290 34
	4	2	1 92	351.30	345.52	345 52
	5	2	1.84	351.08	334.96	334 96

the results presented are the same for permutation flow line scheduling with makespan or weighted sum of completion times performance criteria. An integer programming formulation of the cyclic scheduling problem is presented in Karabati and Kouvelis (1991), and can be extended to include the worker allocation vector as a decision variable; however, the resulting formulation is a difficult nonlinear integer optimization problem. The difficulty of the problem for any realistic size flow line scheduling problem is well outside the limits of the state of the art algorithms for such integer problems.

The interaction between scheduling decisions and allocation of resources is very important in terms of achieving higher throughput rates, and therefore the problem needs to be addressed despite its very difficult nature. The sensitivity results of §5 clearly indicate a high degree of interdependency between these two operational decisions. We suggest below a simple iterative solution procedure to find heuristic solutions for the simultaneous scheduling and discrete resource allocation problem.

Procedure Iterative:

Step 0. Choose an initial schedule σ.

Step 1. Solve (DRAP) for schedule σ.

Step 2. Solve the scheduling problem with the optimal allocation vector X obtained in Step 1.

Step 3. Let γ be the optimal schedule obtained in Step 2. If no improvement has been achieved in the objective function value, stop; otherwise let $\sigma = \gamma$ and go to Step 1.

Table 6 Performance of Procedure Iterative: 10-Job Problems

Problem Class	Problem No	Number of Iter	CPU (sec)	Objective Fn Initial	Objective Fn Final	Objective Fn BM	BM CPU (sec)
C	1	4	29 80	458 48	388 11	451 93	175 25
	2	5	32 30	467 40	388 11	459 83	124 92
	3	5	23 81	407 94	335 83	381 12	201 56
	4	6	37 56	440 02	361 97	410 49	220 46
	5	5	28 45	491 30	426 09	464 94	165 03
D	1	3	10 24	597 01	514 98	567.84	203 45
	2	5	27 86	516 92	435 44	510 32	198 23
	3	3	29 33	523 03	459 30	511.47	199 45
	4	2	26 96	529 70	447.02	509 85	240 28
	5	3	14 93	483 86	426 55	453 00	164 23

Note that σ is a permutation schedule, and in Step 2 of Procedure Iterative (DRAP) is solved after the jobs are fixed according to schedule σ.

Procedure Iterative is guaranteed to converge to a heuristic solution within a finite number of iterations because, upon completion of Steps 1, 2, and 3, the objective function value of the problem is smaller than or equal to that of the previous iteration, and the procedure searches over the finite set of feasible sequences. Note that no sequence can be used twice by the procedure without causing it to stop.

In Tables 5 and 6 we present our computational experience with Procedure Iterative for the cycle time performance criterion. In Table 5 we consider six-job, five-machine, 10-worker problems. The buffer configuration of the flow line is as follows: the buffer capacities between machines 2 and 3, and 3 and 4 are equal to 1, all other buffer capacities are equal to zero. For each problem we report the number of iterations, total CPU time and the initial, i.e., upon completion of Step 1 in the first iteration of the procedure, and final values of the objective function. In the last column of Table 5, we report the optimal objective function value of each problem, which is obtained by solving the resource allocation problem for all possible sequences. We note that this approach is feasible for six-job problems, since we have only 720 permutation schedules. For the problems reported in Table 5, the average gap between the objective function values of the local (i.e., solution generated by Procedure Iterative) and optimal solutions as a percentage of the optimal objective function value has been observed to be equal to 1.84%, demonstrating the very satisfactory performance of the approximate solution procedure. In Table 6, we report our computational experience with 10-job, six-machine, 12-worker problems in classes C and D. The buffer configuration of the flow line is as follows: the buffer capacities between machines 2 and 3, 4 and 5, and 5 and 6 are equal to 1; all other buffer capacities are equal to zero. We have observed very similar results for problem classes A and B; however, we omit them for the brevity of the paper. For the lack of a good lower bound on the optimal objective function value of the simultaneous scheduling and resource allocation problem, however, we have not been able to measure the performance of Procedure Iterative in relative terms for 10-job problems. Instead,

for each problem we randomly created 100 schedules and solved the fixed sequence problem for each of these schedules. The best objective function value obtained by this alternative procedure is reported in the benchmark (BM) column of Table 6, along with the total CPU time requirement of this approach.

The results of Tables 5 and 6 indicate that our approximate solution procedure for the simultaneous scheduling and resource allocation problem is very effective in terms of its computational requirements, and can generate good heuristic solutions.

7. Conclusion

In this paper we have addressed the discrete resource allocation problem in flow lines for various performance criteria. We have shown that the problem is NP-complete in the strong sense, even when the sequence of jobs is fixed. We have developed a very efficient solution procedure for the fixed sequence problem, and presented an extensive computational analysis of its performance on different problem classes. The solution procedure is a branch-and-bound procedure which uses a surrogate relaxation of the problem to generate lower and upper bounds on the optimal value of the objective function. The modeling framework and solution procedure presented in the paper are very flexible, and can be easily used to solve a newly introduced general class of static resource allocation problems in acyclic directed graphs.

The computational experiments suggest that there exists a high degree of interaction between the resource allocation and scheduling decisions. Significant improvements in the performance of the flow line can be achieved when the two decisions are simultaneously addressed. The above fact motivated us to consider the simultaneous resource allocation and scheduling problem. We have developed an approximate and iterative solution procedure which finds very good heuristic solutions with a reasonable amount of computational effort.

We plan to consider, as extensions of this work, the case where workers possess different skill levels (i.e., the problem with nonidentical workers), or there exists a predefined feasible set of worker assignments for each machine in the flow line. The main difficulty associated with handling these extensions within the current model lies in solving (DRAS') with the additional constraints. In addition, the branching rules of our algorithm have to be modified to account for dissimilarities between the workers. Therefore, the approach presented in this paper does not directly apply to the above extensions of the problem. Further research and appropriate algorithmic developments are deemed necessary to handle such extensions.[1]

[1] The authors are grateful to the Associate Editor and two anonymous reviewers for their careful scrutiny of our original draft of the paper and for constructive suggestions which significantly improved the presentation of our results We would also like to thank the Departmental Editor Awi Federgruen for suggesting the use of the greedy procedure as an approximate solution procedure.

Appendix

A. The Finite Capacity Buffers Case

First, without any loss of generality, we may assume that all buffers have either zero capacity or infinite capacity, because we can represent each unit buffer location by a machine on which all processing times are equal to zero We note that when the capacity of a buffer location between two machines is greater than the number of jobs, without any loss of generality, it can be assumed that the buffer capacity between these two machines is infinite Therefore, the transformation of a problem with finite capacity buffers into a problem with zero or infinite capacity buffers is polynomial in the number of jobs and machines. Thus, in order to extend the framework, we need only to find a way to handle the case where the buffer capacity between machines M_j and M_{j+1} is equal to zero. Let us first note that in the case of zero capacity buffers, we refer to the completion time of a job on a machine as the time at which the job leaves the machine, which, due to the blocking phenomenon, might not coincide with the time at which the operation on that job is completed on the machine In order to account for the blocking phenomenon in our graph theoretic model (see Figure 1), we define diagonal arcs from each vertex $(i, j + 1)$ to vertex $(i + 1, j)$, $i = 1, 2, \ldots, n$ This modeling approach allows $C_{i,j}$ to be found by determining the weight of the maximum-weighted path form vertex $(1, 1)$ to vertex (i, j). However, in computing the weight of a path between these two nodes, the weight of a vertex which is entered via one of the diagonal arcs is taken to be zero, as is required by the blocking phenomenon. We also note that the problem with finite capacity buffers can be treated as a special case of the discrete resource allocation problem in acyclic directed graphs which is discussed in §3

B. Proof of Theorem 1

We reduce the NP-complete Set Covering Problem (SCP) (see Papadimitriou and Steiglitz 1982) to (DRAP). Define the set-element incidence matrix for (SCP) as $a_{ij} = 1$, if element i is covered by (included in) set S_j; 0 otherwise, for $i = 1, 2, \ldots, n; j = 1, 2, \ldots, m$. For a given integer $N > 0$, (SCP) tries to answer the question whether there exists a solution $Y = (y_1, y_2, \ldots, y_m)$ with

$$\sum_{j=1}^{m} y_j \le N,$$

$$\sum_{j=1}^{m} a_{ij} y_j \ge 1, \quad i = 1, 2, \ldots, n,$$

$$y_j \in \{0, 1, 2, \ldots, N\}, \quad j = 1, \ldots, m.$$

Note that the extension of the solution space from binary integers to general integers will not change the yes/no answer to the problem.

For a given instance of (SCP) we define the following reductions.

B.1. The Makespan Performance Criterion. For a given (SCP) we define a $(m + 2)$-machine n-job scheduling problem with infinite capacity buffers and the following processing times:

$$p_{i,1}(x_1) = K, \; p_{i,m+2}(x_{m+2}) = K, \quad i = 1, 2, \ldots, n,$$

$$p_{i,j}(x_j) = L - a_{i,j-1}x_j, \quad i = 1, 2, \ldots, n; \quad j = 2, 3, \ldots, m + 1,$$

where K and L are positive constants, and $K \gg nL$, $L \gg N \max_{i,j} a_{i,j}$. Note that all processing times are nonincreasing and linear (convex) in x_j, $j = 1, 2, \ldots, m + 2$.

Note that this is a problem with two bottleneck machines, M_1 and M_{m+2} (i e , $\min_{1\le i\le n} p_{i,1} \ge \sum_{i=1}^{n} p_{i,j}$, $j = 2, 3, \ldots, m + 1$, and $\min_{1\le i\le n} p_{i,m+2} \ge \sum_{i=1}^{n} p_{i,j}$, $j = 2, 3, \ldots, m + 1$), and the number of dominating paths from vertex (1, 1) to vertex $(n, m + 2)$ is equal to n (see Monma and Rinnooy Kan 1983 for a detailed discussion of bottleneck machine concept). These paths have the following form.

$$\tau_i = ((1, 1), (2, 1), \ldots, (i, 1), (i, 2), \ldots, (i, m + 2), (i + 1, m + 2), (i + 2, m + 2), \ldots, (n, m + 2)),$$

with a weight of

$$f(\tau_i, X) = (n + 1)K + mL - \sum_{j=2}^{m+1} a_{i,j-1}x_j, \quad i = 1, 2, \ldots, n.$$

We now write the following (DRAP):

$$\text{(DRAP)} \quad f_{\text{DRAP}} = \min M$$

$$\text{subject to} \quad (n + 1)K + mL - \sum_{j=2}^{m+1} a_{i,j-1}x_j \le M, \quad i = 1, 2, \ldots, n,$$

$$\sum_{j=1}^{m+2} x_j \le N,$$

$$x_j \in \{0, 1, 2, \ldots, N\}, \quad j = 1, \ldots, m + 2$$

Let X^* be an optimal solution to the above (DRAP) Then $x_1^* = 0$ and $x_{m+2}^* = 0$, because processing times on machines M_1 and M_{m+2} are constant, and not a function of the number of resources allocated to these machines In order to complete the reduction from (SCP), we now ask the question whether there exists a solution to (DRAP) with $f_{\text{DRAP}} \le (n + 1)K + mL - 1$. Therefore, if the answer to above question is *yes*, then the answer to (SCP) is also *yes*, with $y_j = x_{j+1}$, $j = 1, 2, \ldots, m$

B.2. The Weighted Sum of Completion Times Performance Criterion. The makespan criterion is a special case of the weighted sum of completion times criterion with $r_i = 0$, $i = 1, 2, \ldots, n - 1$, and $r_n = 1$; therefore (DRAP) is NP-complete for the weighted sum of completion times criterion as well.

B.3. The Cycle Time Performance Criterion. We define a $(2n + m)$-machine $(n + m + 1)$-job problem, with zero capacity buffers between the machines The processing times are defined as follows.

$$p_{i,n+1-i}(x_{n+1-i}) = iK, \quad i = 1, 2, \ldots, n,$$

$$p_{i,2n+m+1-i}(x_{2n+m+1-i}) = (n - i)K, \quad i = 1, 2, \ldots, n,$$

$$p_{i,n+j}(x_{n+j}) = L - a_{i,j}x_{n+j}, \quad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, m,$$

and all other processing times are equal to a constant p, where $K \gg L$, $L \gg N \max_{i,j} a_{i,j}$, and $L \gg p$

In a cyclic scheduling problem with $(n + m + 1)$ jobs there are exactly $(n + m + 1)$ operations on each path (see Karabati and Kouvelis 1991) In this instance of the problem, due to the special structure of the processing times, there are n dominating paths with the following weights·

$$f(\tau_i, X) = nK + (n - 1)p + mL - \sum_{j=1}^{m} a_{i,j}x_{n+j}, \quad i = 1, 2, \ldots, n.$$

We now write the following (DRAP):

$$\text{(DRAP)}: \quad f_{\text{DRAP}} = \min M,$$

$$\text{subject to} \quad nK + (n - 1)p + mL - \sum_{j=1}^{m} a_{i,j}x_{n+j} \le M, \quad i = 1, 2, \ldots, n,$$

$$\sum_{j=1}^{2n+m} x_j \le N,$$

$$x_j \in \{0, 1, 2, \ldots, N\}, \quad j = 1, \ldots, 2n + m.$$

Let X^* be an optimal solution to the above (DRAP). Then $x_i^* = 0$, and $x_{n+m+i}^* = 0$, $i = 1, 2, \ldots, n$, because processing times on machines M_i and M_{n+m+i}, $i = 1, 2, \ldots, n$, are constant and not a function of the resources allocated. In order to complete the reduction from (SCP), we now ask whether there exists a solution to (DRAP) with $f_{\text{DRAP}} \le nK + (n - 1)p + mL - 1$ Therefore, if the answer to the above question is *yes*, then the answer to (SCP) is also *yes*, with $y_j = x_{n+j}$, $j = 1, 2, \ldots, m$. □

References

Bazaraa, M. S. and C M. Shetty, *Nonlinear Programming: Theory and Algorithms*, John Wiley & Sons, New York, 1979

Bitran, G. R and A. C. Hax, "Disaggregation and Resource Allocation Using Convex Knapsack Problems with Bounded Variables," *Management Sci* , 27 (1981), 431–441

Daniels, R. L. and J. B. Mazzola, "Flow Shop Scheduling with Resource Flexibility," *Oper. Res.*, 42, 3 (1994), 504–522.

Federgruen, A and H Groenevelt, "The Greedy Procedure for Resource Allocation Problems: Necessary and Sufficient Conditions for Optimality," *Oper Res.*, 34, 6 (1986), 909–918.

Ibaraki, T and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches*, The MIT Press, Cambridge, MA, 1988.

Karabati, S. and P. Kouvelis, "Cyclic Scheduling in Flow Lines: Mod-

eling Observations, Effective Heuristics and an Optimal Solution Procedure," Naval Res. Logist., forthcoming

Lawler, E. L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

Luss, H. and S K. Gupta, "Allocation of Effort Resources Among Competitive Activities," *Oper. Res.*, 23 (1975), 360–366.

McCormick, S. T., M. L. Pinedo, and B. Wolf, "The Complexity of Scheduling a Flexible Assembly System," Working Paper, University of British Columbia, Vancouver, BC, Canada, 1987.

——, S. Shenker, B. Wolf, and M. L Pinedo, "Transient Behavior in a Flexible Assembly System," *International J. Flexible Manufacturing Systems*, 3 (1990), 27–44.

Moder, J. J., C C. Phillips, and E. W. Davis, *Project Management with CPM and Precedence Diagramming*, Van Nostrand Rinehart, New York, 1970

Monma, C L. and A. H. G Rinnooy Kan, "A Concise Survey of Efficiently Solvable Special Cases of the Permutation Flowshop Problem," *RAIRO*, 17 (1983), 105–119.

Papadimitriou, C. H. and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

Wiest, J. and F. Levy, *A Management Guide to PERT/CPM, with GERT/PDM/DCPM and Other Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1977.

Accepted by Awi Federgruen; received August 7, 1992. This paper has been with the authors $5\frac{1}{2}$ *months for 3 revisions.*