A PARALLEL MULTI-OPERATION SCHEDULING PROBLEM WITH MACHINE ORDER CONSTRAINTS

Weizhen Mao
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187-8795
wm@cs.wm.edu

KEYWORDS

Job scheduling, parallel processing, network routing, on-line algorithms.

ABSTRACT

In this paper, we define a multi-operation job scheduling problem with parallel machines. What makes this problem unique is the added machine order constraints, which require that only certain machine orders be used for the execution of jobs. We study the complexity of the problem and analyze two on-line greedy heuristics. In addition, we explore the relation of our scheduling problem to a network routing problem.

1 INTRODUCTION

The scheduling problem we consider in this paper belongs to the class of *multi-operation scheduling problems* [7], in which the execution of a job includes multiple operations to be performed by machines of different types. These scheduling problems often arise in industrial manufacturing, production planning, and computer control [1]. For example, consider a large automotive garage with specialized workshops [5]. A car may require the following work: replace exhaust system, align wheels, and tune up. Since the three corresponding workshops are in different buildings, it is impossible to perform two operations for a car simultaneously. When there are many cars requiring services at the automotive garage, it is desirable to construct a service schedule that takes the least amount of total time. To distinguish our problem from the classical multi-operation scheduling problems, we call ours the *parallel multi-operation scheduling problem*, where parallel machines, instead of a single machine, are available to perform a certain type of operation.

In our parallel multi-operation scheduling problem, there are m *types* of machines, $T_1, \ldots, T_m$, providing services to n independent *jobs*, $J_1, \ldots, J_n$. In the parallel processing environment, each set T_i contains k_i *machines*, $M_{i1}, \ldots, M_{ik_i}$, of the same type and probably with different speeds (although in this paper we assume that all machines of the same type are identical). For each job J_j, its execution consists of m operations to be performed by one machine of each type, in the operation order of $T_1, \ldots, T_m$. In other words, job J_j will be executed by machine M_{1j_1} of type T_1, then by machine M_{2j_2} of type T_2, etc., and finally by machine M_{mj_m} of type T_m. The entire execution of a job must be sequential and strictly in the given order. However, it is up to the scheduling algorithm to decide which machine M_{ij_i} of type T_i to use for the ith operation, but theoretically, j_i can be any integer between 1 and k_i. Let $p[J_j, T_i, M_{il}]$ be the *processing (execution) time* of job J_j on the lth machine M_{il} of type T_i, for $1 \leq j \leq n$, $1 \leq i \leq m$, and $1 \leq l \leq k_i$. For notational simplicity, we will sometimes use $p[j, i, l]$ for the same definition.

So far, the problem described is no different from the previously studied *hybrid flow shop scheduling* [6], which is a multi-operation scheduling problem with parallel machines and a fixed operation order to be followed by all jobs. However, what makes our problem different and likely more difficult is the so-called *machine order constraints* imposed on the problem instances, which arise in situations where some machines of different types may be geographically isolated or mutually incompatible so that they can not be chosen simultaneously for the execution of one job. For example, if job J_1 chooses machine M_{11} of type T_1 for its first operation, and M_{11} and machine M_{23} of type T_2 are not compatible to be used by the execution of the same job, then job J_1 is not allowed to use M_{23} for its second operation. To enforce the constraints, we define a set of *feasible machine orders*, called F, where each element in F is an m-tuple, representing one possible machine order that may be used for the execution of jobs. Set F is given as a parameter of the problem instances.

We define the above scheduling problem as an optimization problem by employing the *makespan* of a schedule as the objective function. The goal is to construct a schedule with the minimum makespan, i.e., the minimum total elapsed time from the start of the first job to the completion of the last job in a schedule.

We organize this paper as follows. In Section 2, we give an NP-completeness proof of the problem. In Section 3, we define and analyze two on-line heuristics based on greedy strategies. In Section 4, we explore the relation of the scheduling problem to network routing. Finally, in Section 5, we make our conclusions.

2 COMPUTATIONAL COMPLEXITY

In this section, we restrict our attention to the decision problem, i.e., given a bound B, is there a solution to the parallel multi-operation scheduling problem with the makespan no larger than B? In particular, we consider several special cases of the problem.

First, we observe that when there is only one machine of each type, i.e., $k_i = 1$ for $1 \leq i \leq m$, and $F = \{(M_{11}, M_{21}, \ldots, M_{m1})\}$, the problem degenerates to the classical NP-complete flowshop scheduling problem [4]. Secondly, when $m = 1$ and $F = \{(M_{11}), (M_{12}), \ldots, (M_{1k_1})\}$, the problem then becomes the single-operation multi-machine scheduling problem, which is also NP-complete [3]. Next, we assume that the processing time of any job on each machine is either 1 or ∞, i.e., $p[j,i,l]$ is 1 or ∞, for $1 \leq j \leq n$, $1 \leq i \leq m$, and $1 \leq l \leq k_i$. Since its NP-completeness is not as obvious as the previous two special cases, we provide the proof in the following theorem.

THEOREM 1 *The parallel multi-operation scheduling problem with machine order constraints is NP-complete even when the processing time of any job on each machine is either* 1 *or* ∞.

Proof Consider the NP-complete *timetable design* [2]. Given a set of three work hours $H = \{h_1, h_2, h_3\}$, a set of craftsmen $C = \{c_1, c_2, \ldots, c_m\}$, and a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$. Let $A(c) \subseteq H$ be the available hours for $c \in C$, $A(t) = H$ be the available hours for $t \in T$, and $R(c,t) = 0$ or 1 be the required work hour for $c \in C$ on $t \in T$. The problem asks to define a function $f : C \times H \times T \rightarrow \{0,1\}$ such that (1) $f(c,h,t) = 1$ only if $h \in A(c)$; (2) $\forall c$, $\forall h$, there is at most one t for which $f(c,h,t) = 1$; (3) $\forall h$, $\forall t$, there is at most one c for which $f(c,h,t) = 1$; and (4) $\forall c$, $\forall t$, there are exactly $R(c,t)$ values of h for which $f(c,h,t) = 1$.

Let us define an instance for our scheduling problem. Assume there are two types of machines, in T_1 and T_2, respectively. Each machine in T_1 is represented by a craftsman-hour pair and $T_1 = \{(c,h) | h \in A(c)\}$. Each machine in T_2 is represented by an hour-task pair and $T_2 = \{(h,t) | h \in A(t)\}$. For the feasible machine orders, let $F = \{(p,q) | p = (c,h) \in T_1, q = (h,t) \in T_2\}$, i.e., each element (pair/order) in F consists of two machines whose representations share the same work hour. For each $R(c,t) = 1$, we create a job, represented by the corresponding craftsman-task pair (c,t). For the processing times of the jobs, define $p[j,1,l] = 1$ if job $j = (c,t)$ and machine $l = (c,h)$ for some c,h,t, define $p[j,2,l] = 1$ if job $j = (c,t)$ and machine $l = (h,t)$ for some c,h,t, and define $p[j,i,l] = \infty$ otherwise. Finally, let the bound in the decision problem be 2.

We claim that the timetable design problem has a solution f satisfying requirements (1)-(4) if and only if there is a schedule with makespan 2 for the instance of the parallel multi-operation scheduling problem.

Assume that there is $f : C \times H \times T \rightarrow \{0,1\}$ satisfying requirements (1)–(4) in the timetable design problem. Consider all triples (c,h,t) with $f(c,h,t) = 1$, each of which corresponds to a machine order $((c,h),(h,t))$, where (c,h) is a machine in T_1 and (h,t) is a machine in T_2. We use these machine orders to form a set, F'. Obviously, $F' \subseteq F$ (due to requirement (1) and the definition of F). Because of requirements (2) and (3), there is no machine appearing in more than one feasible machine order in F'. Furthermore, because of requirement (4), for each job (c,t), there is exactly one machine order, $((c,h),(h,t))$, in F' which yields a finite total processing time. This means that all jobs can be executed in parallel starting at time 0 and finishing at time 2. Therefore, we have a schedule with makespan 2.

Assume that there is a schedule with makespan 2 for the instance of the parallel multi-operation scheduling problem. Then there must be machine-disjoint machine orders in F, one for each job. We define f such that $f(c,h,t) = 1$ if and only if machine order $((c,h),(h,t))$ in F is used to execute job (c,t). It is easy to verify that requirements (1)-(4) are all satisfied. Therefore, the timetable design problem has a solution.

Since the parallel multi-operation scheduling problem is in NP and a polynomial reduction from an NP-complete problem to the scheduling problem exists, we conclude that the parallel multi-operation scheduling problem with processing times of 1 or ∞ is NP-complete. □

The NP-completeness of the scheduling problem indicates that no polynomial-time algorithm has been designed to construct optimal schedules for the problem and thus research should focus on designing fast algorithms that give near-optimal solutions.

3 ON-LINE ALGORITHMS

In a realistic situation, jobs are not available all at once before any scheduling decision is made, but instead they arrive one after another in a sequence $J_1, \ldots, J_n$ and have to be processed in the order given. We call algorithms that are able to process jobs one by one without knowledge of future jobs *on-line algorithms*.

Competitive analysis is commonly used to judge the quality of solutions produced by an on-line algorithm. Let A be the on-line algorithm under consideration and OPT be the optimal (off-line) algorithm for the same problem. Furthermore, let $C_A(I)$ and $C_{OPT}(I)$ be the values of the objective function (makespan) when algorithms A and OPT are applied to instance I, respectively. If there are c (a constant or a function of the size of I) and a (a constant), such that $C_A(I) \leq c \cdot C_{OPT}(I) + a$ for any I, then we say that on-line algorithm A is *c-competitive*, where c is its *competitive ratio*. The c-competitiveness of an algorithm indicates that in the worst case the value of the objective function of a solution produced by the on-line algorithm is at most c times that of the optimal solution.

Next, we study two on-line algorithms, GREEDY1 and GREEDY2, for our scheduling problem, both based

on some greedy strategies. The first algorithm uses a naive greedy strategy, in which for each job J_j ($j = 1, \ldots, n$), the algorithm picks an arbitrary feasible machine order from F that yields a finite total processing time for J_j, and schedules the job on those machines in a way that gives the smallest increase in the current makespan of the schedule. In other words, the algorithm tries to balance the load of each machine at any time of the schedule. We call this algorithm GREEDY1.

Since the algorithm chooses a feasible machine order arbitrarily, it can not be avoided that some machines be used heavily in some instances. As a matter of fact, in the worst case we can force one particular machine order to be used over and over again by the execution of all the jobs.

THEOREM **2** *GREEDY1 is at least $\Theta(n)$-competitive, where n is the number of jobs.*

Proof Consider an instance I, where in each T_i there are n machines, i.e., $k_i = n$ for $1 \leq i \leq m$. Assume that each of the m operations of a job takes unit time by any machine of the corresponding type, i.e., $p[j, i, l] = 1$ for $1 \leq j \leq n$, $1 \leq i \leq m$, and $1 \leq l \leq n$. Also, let $F = \{(M_{1l}, M_{2l}, \ldots, M_{ml}) | 1 \leq l \leq n\}$. Clearly, job J_j can use machine order $(M_{1j}, M_{2j}, \ldots, M_{mj})$ for its execution, and all jobs can be processed in parallel from time 0 to time m. So the optimal makespan $C_{OPT}(I) = m$. See Figure 1.

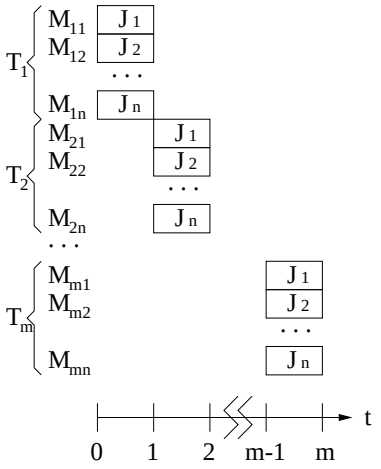

Figure 1. The optimal schedule.

As for the schedule produced by GREEDY1, since machine orders are chosen arbitrarily from F, in the worst case all jobs will be using the same machine order, say $(M_{11}, M_{21}, \ldots, M_{m1})$. Therefore, jobs will line up for the service of a single machine. It is easy to figure out that $C_{GREEDY1}(I) = m + n - 1$. See Figure 2.

The ratio $\frac{C_{GREEDY1}(I)}{C_{OPT}(I)} = \frac{m+n-1}{m} = \Theta(n)$, when m is a constant. Since we have found one instance (described above) which yields a ratio of $\Theta(n)$, then the worst-case ratio among all instances must be at least $\Theta(n)$. Therefore, $GREEDY1$ is at least $\Theta(n)$-competitive. □

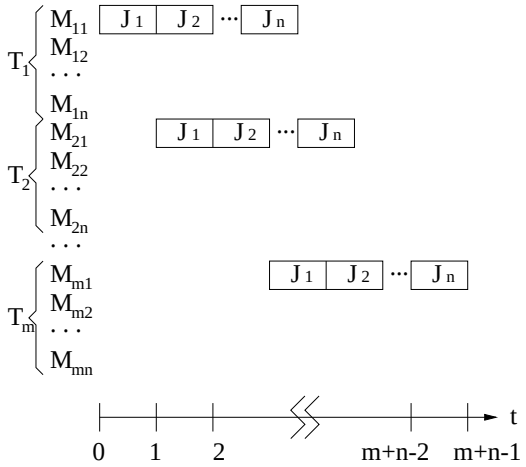

Figure 2. The GREEDY1 schedule.

To improve the performance of a greedy algorithm, we have to adopt a smart strategy to select machine orders to spread the processing load among all machines. Here we propose another greedy algorithm, GREEDY2, where we define $\mu(M)$ to be the number of jobs that have used machine M for their operations so far into the schedule. For each job that comes in the on-line sequence, GREEDY2 always selects a feasible machine order that causes the smallest increase, if at all, in $\max_{\forall M}\{\mu(M)\}$, also denoted by μ_{max}.

THEOREM **3** *GREEDY2 is at least $\Theta(\sqrt{n})$-competitive, where n is the number of jobs.*

Proof We consider an instance I with n on-line jobs such that $n = 1+2+\cdots+h = \frac{1}{2}h(h+1)$ for some integer h. So $h = \Theta(\sqrt{n})$. The jobs are denoted by J_{il} for $1 \leq i \leq h$ and $1 \leq l \leq i$. Assume that there are two types of machines, i.e., $m = 2$. Let $T_1 = \{A_{il} | 1 \leq i \leq h \text{ and } 1 \leq l \leq i\}$ and $T_2 = \{B_{il} | 1 \leq i \leq h \text{ and } 1 \leq l \leq 2i-1\}$. The set of feasible machine orders, F, is the union of subsets, $F_1, \ldots, F_h$, where for any $1 \leq i \leq h$, $F_i = \{(A_{i1}, B_{i1}), (A_{i1}, B_{i2}), \ldots, (A_{i1}, B_{ii})\} \cup \{(A_{i2}, B_{i(i+1)}), (A_{i3}, B_{i(i+2)}), \ldots, (A_{ii}, B_{i(2i-1)})\}$. We define the processing times in such a way that jobs $J_{i1}, J_{i2}, \ldots, J_{ii}$ have no choice but to use the machine orders in F_i. That is, let $p[J_{il}, T_1, A_{i1}] = 1$ for $1 \leq l \leq i$, $p[J_{il}, T_1, A_{i(l+1)}] = 1$ for $1 \leq l \leq i-1$, $p[J_{il}, T_2, B_{il}] = 1$ for $1 \leq l \leq i$, and $p[J_{il}, T_2, B_{i(i+l)}] = 1$ for $1 \leq l \leq i-1$. Set the rest of $p[*, *, *]$ to be ∞.

Let us consider the optimal schedule. For job J_{il}, $1 \leq l \leq i-1$, machine order $(A_{i(l+1)}, B_{i(i+l)})$ is chosen. For job J_{ii}, machine order (A_{i1}, B_{ii}) is chosen. Since no machine is used by more than one job, all jobs can be executed in parallel from time 0 to time 2 (one unit for each operation). Therefore, $C_{OPT}(I) = 2$.

Now for the greedy algorithm GREEDY2, first initialize μ to be 0 for all machines. Assume that jobs are ordered $J_{11}, J_{21}, J_{22}, J_{31}, J_{32}, J_{33}, \ldots$. For job J_{11}, since there is only one feasible machine order (A_{11}, B_{11}), GREEDY2 picks it and assigns the job on machines A_{11} and B_{11}, forcing $\mu(A_{11}) = \mu(B_{11}) = 1$, and hence $\mu_{max} = 1$. Clearly, the current makespan is 2. For the next job J_{21} in the sequence, GREEDY2 has two choices, machine orders (A_{21}, B_{21}) and (A_{22}, B_{23}). Since

both can be used for the execution of job J_{21} and neither causes any increase in μ_{max}, GREEDY2 pick an arbitrary machine order. Assume it is (A_{21}, B_{21}). Then job J_{22} comes. Since only machine order (A_{21}, B_{22}) is feasible for the job, GREEDY2 has no choice but to select (A_{21}, B_{22}), forcing machine A_{21} to be used by two jobs, thus $\mu_{max} = 2$. The current makespan is 3 after the first three jobs are scheduled. Next consider jobs J_{31}, J_{32}, J_{33}. If GREEDY2 happens to choose machine order (A_{31}, B_{31}) for job J_{31} (this is certainly a possibility since the selection does not change μ_{max}), then for the next two jobs J_{32} and J_{33}, the algorithm has no choice but to pick machine orders (A_{31}, B_{32}) and (A_{31}, B_{33}), respectively. Thus $\mu_{max} = 3$ and the makespan becomes 4. We continue this reasoning until the last group of jobs $J_{h1}, J_{h2}, \ldots, J_{hh}$ are scheduled. We can see that the final value of μ_{max} is h and the makespan of the schedule is $h + 1$. So $C_{GREEDY2}(I) = h + 1$.

The ratio $\frac{C_{GREEDY2}(I)}{C_{OPT}(I)} = \frac{h+1}{2} = \Theta(\sqrt{n})$. Since we have found one instance (described above) which yields a ratio of $\Theta(\sqrt{n})$, then the worst-case ratio among all instances must be at least $\Theta(\sqrt{n})$. Therefore, the competitive ratio of GREEDY2 is at least $\Theta(\sqrt{n})$. □

We are unable to prove the tight competitive ratios for GREEDY1 and GREEDY2, but preliminary experiments show that on average GREEDY2 performs much better than GREEDY1, i.e., the makespan of the schedule constructed by GREEDY2 is much shorter than that of the schedule constructed by GREEDY1.

4 A RELATED NETWORK ROUTING PROBLEM

Let us consider a *network routing problem* [8] [9]. Define a *layer graph* $G = (V_0 \cup V_1 \cup \cdots \cup V_m, E_1 \cup E_2 \cup \cdots \cup E_m)$, where $V_0, V_1, \ldots, V_m$ are disjoint sets of vertices and E_i is the set of directed edges between vertices in V_{i-1} and V_i. See Figure 3 for an example.

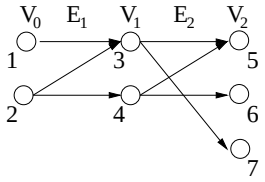

Figure 3. A layer graph with $m = 2$.

In the network routing problem, the underlying network is described as a layer graph, where vertices represent processors and edges represent message-passing links. Assume that n files are to be transferred from their pre-specified sources in V_0 to their pre-specified destinations in V_m. The network routing problem asks to assign to each file a route between its source and destination and construct a schedule for each file so that the location of the file is accounted for at each instant of time and network resources are not over-utilized (assuming that at any time only one file can be traversing a link), with the goal to minimize the overall makespan. Using the link speed and file length, it is easy to compute the time needed for a file to traverse a certain link in the network.

We show in this section that the above network routing problem can be converted to our parallel multi-operation scheduling problem. However, as we shall see later, the conversion does not hold true vice versa.

Given a network graph with $m + 1$ vertical layers of nodes, with the leftmost layer to be the possible source nodes and the rightmost layer to be the possible destination nodes. For each edge in E_i, we define a machine of type T_i. So the number of parallel machines of type T_i is $|E_i|$, and there are m types of machines altogether. Each path from a source (in V_0) to a destination (in V_m) corresponds a feasible machine order for the execution of jobs. For example, if edges $(1, 3)$ and $(3, 5)$ in a layer graph with $m = 2$ represent machines M_{11} and M_{21}, respectively, and there is a path $1 \to 3 \to 5$ between source node 1 and destination node 5, then machine order (M_{11}, M_{21}) is a member of the set of feasible machine orders, F. For each file with source s_j and destination d_j, we create a job J_j which has to go through m operations. As for the processing time, we use the following principle: If job J_j is created from a file with file length h and machine M_{il} of type T_i corresponds to an edge with speed t, then $p[J_j, T_i, M_{il}]$, the processing time of job J_j on machine M_{il} of type T_i, is defined to be $\frac{h}{t}$. However, there is one exception to this principle. If the machine's corresponding edge is either the first or last edge on a path, then assign ∞ to $p[J_j, T_i, M_{il}]$ for the case when the nodes on the edge do not match the source s_j or the destination d_j. This prevents a job from selecting a machine order when the corresponding path in the network can not be used to transfer the file from its source to destination. For example, if edge (a, b) with $a \in V_0$ corresponds to machine M_{11}, and file f_1 with source $c \neq a$ corresponds to job J_1, then $p[J_1, T_1, M_{11}] = \infty$.

Assume the graph in Figure 3 is used to transfer 3 unit-length files. For simplicity, assume the link speed is also unit. Let file $f_1 = (1, 5)$, where node 1 is its source and node 5 is its destination, file $f_2 = (2, 5)$, and file $f_3 = (1, 7)$. The following tables describe the corresponding instance for the parallel multi-operation scheduling problem. The first table shows the edge-to-machine correspondence. The second table shows the path-to-order correspondence. The third table defines the processing times of the jobs.

edge	machine
$(1,3)$	M_{11}
$(2,3)$	M_{12}
$(2,4)$	M_{13}
$(3,5)$	M_{21}
$(3,7)$	M_{22}
$(4,5)$	M_{23}
$(4,6)$	M_{24}

Path	Machine order
$1 \to 3 \to 5$	(M_{11}, M_{21})
$1 \to 3 \to 7$	(M_{11}, M_{22})
$2 \to 3 \to 5$	(M_{12}, M_{21})
$2 \to 3 \to 7$	(M_{12}, M_{22})
$2 \to 4 \to 5$	(M_{13}, M_{23})
$2 \to 4 \to 6$	(M_{13}, M_{24})

p	J_1	J_2	J_3
M_{11}	1	∞	1
M_{12}	∞	1	∞
M_{13}	∞	1	∞
M_{21}	1	1	∞
M_{22}	∞	∞	1
M_{23}	1	1	∞
M_{24}	∞	∞	∞

We can show that there is a solution to an instance of the network routing problem with layer graphs if and only if there is a solution to the defined instance of the parallel multi-operation scheduling problem with machine order constraints. The formal proof is omitted in this paper. In this one-way conversion, files are mapped to jobs, edges are mapped to machines of various types, and paths are mapped to feasible machine orders. However, the conversion does not hold true in the opposite direction. That is, given any instance of the parallel multi-operation scheduling problem with machine order constraints there is not always an equivalent instance of the network routing problem with layer graphs. The reason for this claim is that feasible machine orders do not necessarily define paths in layer graphs. For example, assume $m = 2$ and $F = \{(M_{11}, M_{21}), (M_{11}, M_{22}), (M_{12}, M_{21})\}$. Creating an edge for each machine, we get a network graph as shown in Figure 4. While for each feasible machine order there is a corresponding path in the graph (e.g., for machine order (M_{11}, M_{21}), it is path $1 \rightarrow 3 \rightarrow 4$), there exists a path, $2 \rightarrow 3 \rightarrow 5$, for which there is not a corresponding feasible machine order since $(M_{12}, M_{22}) \notin F$. So the feasible machine order set F does not uniquely define the set of all paths in the network graph. Hence, the conversion from the scheduling problem to the routing problem does not hold true.

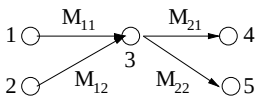

Figure 4. A network graph.

5 CONCLUSIONS

In this paper, we defined a scheduling problem, in which parallel machines of various types are available to process jobs consisting of multiple operations. What makes our problem different from all of the previously studied scheduling problems is the added machine order constraints. We provided the NP-completeness proof for the problem, and designed and analyzed two on-line greedy algorithms for the problem. We also gave our observation that a network routing problem can be converted to our scheduling problem. However, the conversion does not stand vice versa.

For future research, we are interested in proving the tight competitive ratios for both greedy algorithms. We also wish to explore further the relation between the scheduling problem and the routing problem. We suspect that many results developed for the network routing problem may be applied to the scheduling problem.

ACKNOWLEDGEMENT

This research is supported in part by NSF grant NCR-9505963.

References

[1] Coffman, E. G. Jr.(1976), *Computer and Job Shop Scheduling Theory*, John Wiley and Sons, New York.

[2] Evan, S., Itai, A., and Shamir, A. (1976), On the complexity of timetable and multicommodity flow problems, *SIAM Journal on Computing*, **5**, 691–703.

[3] Garey, M. R., Johnson, D. S. (1979), *Computers and Intractability: A guide to the theory of NP-completeness*, Freeman, San Francisco.

[4] Garey, M. R., Johnson, D. S., and Sethi, R. (1976), The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, **1**, 117–129.

[5] Gonzalez, T. and Sahni, S. (1976), Open shop scheduling to minimize finish time, *Journal of ACM*, **23**, 665–679.

[6] Gupta, J. N. D. (1988), Two-stage hybrid flowshop scheduling problem, *Journal of the Operational Research Society*, **39**, 359–364.

[7] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B. (1990), Sequencing and scheduling: Algorithms and complexity, *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan and P. Zipkin, ed., North-Holland.

[8] Mao, W. and Simha, R. (1994), Routing and scheduling file transfers in packet-switched networks. *Journal of Computing and Information*, **1**, Special Issue: *Proceedings of the 6th International Conference on Computing and Information*, 559–574.

[9] Rivera-Vega, P. I., Varadarajan, R., and Navathe, S. B. (1990), Scheduling data redistribution in distributed databases. *Proceedings of the 6th International Conference on Data Engineering*, 166-173.

AUTHOR'S BIOGRAPHY

Weizhen Mao received her Ph.D. in computer science from Princeton University in 1990. She then joined the faculty of the College of William and Mary and is now Associate Professor in the Department of Computer Science. Her research interests are in design and analysis of algorithms and combinatorial optimization. She has published papers in the areas of bin packing, job scheduling, network routing, and graph partitioning and mapping.