〈Research Paper〉

Journal of the Korean Institute of Industrial Engineers
Vol. 1 No.2 December 1975

An Improved Branch-and-Bound Algorithm for Scheduling Jobs on Identical Machines

Sung Hyun Park*

Abstract

In an earlier paper ('Scheduling Jobs on a Number of Identical Machines' by Elmaghraby and Park, March 1974, AIIE Transactions) a branch-and-bound algorithm was developed for the sequencing problem when all jobs are available to process at time zero and are independet (i.e., there are not a priori precedence relationships among jobs.). However, the amount of computation required by the algorithm was not considered to be short if more than 50 jobs were processed. As an effort to improve the algorithm, the present paper modifies the implicit enumeration procedure in the algorithm so that moderately large problems can be treated with what appears to be a short computational time. Mainly this paper is concerned with improving the lower bound in the implicit enumeration procedure. The computational experiences with this new branch-and-bound algorithm are given.

1. Introduction

The problem to be discussed in this paper is that of finding a schedule of N jobs (tasks, activities, etc.) on M identical machines (processors, facilities, etc.) in a one-stage production process. Let L_j denote the lateness of job j, defined as

$$L_j = \max\ (O;\ T_j - d_j)$$

where d_j is the due date of job j and T_j is the time of completion for the particular schedule being evaluated. Then the objective is to minimize the total penalty cost of lateness of jobs as measured by

$$\sum_{j=1}^{N} p_j L_j$$

where p_j is the penalty per unit time of lateness in job j.

The problem of sequencing N jobs on M identical machines in a single stage production process arises quite often in various activities of everyday life, for instance: in hospitals (e.g., the sequencing of patients on identical test facilities), in grocery stores (e.g., the sequencing of customers on clerks), in manufacturing shops (e.g., the sequencing of items on lathes), and others. For the sequencing problem the following assumptions are usually made, and, unless otherwise stated, are assumed in the remainder of this paper.

(1) Set-up and processing times are independent of the order in which the jobs are scheduled, and the set-up times are included in the processing times.

(2) Each machine begins operation at time zero and operates without idle time on one job at a time, until all jobs assigned to it are completed.

(3) Each job, once started, must be performed to completion (no job cancellation).

The problems of scheduling jobs on ma-

* Mississippi State University

chines in parallel have been treated by several authors. Notable among these are the papers by Arthanari & Ramamurthy [2], East man, Even & Isaacs [4], Elmaghraby & Park [6], McNaughton [8], Nabeshima[9, 10], Park[11], Root[13] and Rothkopf[14]. References[6] and[11] are particularly relevant to this discussion simply because this is an improved version of the branch-and-bound algorithm developed in the references. Other authors have shown interest in the problem indirectly by their papers on the problem of project planning under constrained resources; see, e.g., the papers of Elmaghraby [5], Mason & Moodie[7], Pritsker, Watters & Wolfe[12] and Schrage[15].

2. Theoretical Background

As mentioned earlier, this paper is concerned with an improved lower bound for the algorithm presented in [6]. Therefore, in order to discuss a modified method, the previous one should be explained. The following is a brief sketch of the theoretical developments appearing in the paper[6] on which the algorithm is theoretically based. The proofs are not given here. The interested readers may consult reference[6].

Let

t_i : processing time for job i,
d_i : due time for job i,
p_i : penalty per unit time of lateness in job i,
r_i : t_i/p_i, the ratio of processing time to the penalty rate,
$Q:(i,j,\cdots,v)$: a sequence in which job i is first, job j next$\cdots$,and job v last,
Q_m : $(m_1, m_2, \cdots, m_k)$ a sequence of k jobs on machine m; $m=1, 2, \cdots, M$,
subscript m_j: the jth job on machine m,
s_i : start time of job i,
T_i : time of completion of job i in a given sequence.

The following particular assumptions are introduced and theoretical developments based on these assumptions are presented.

Assumptions

1. $d_j = t_j$ for job j, *i.e.*, each job is allowed only its processing time before incurring penalty.

2. The jobs are numbered in order of nondecreasing ratios of r_i; breaking ties by placing the job with the smaller t_i first. Therefore $i<j$ implies $r_i \leq r_j$. This order will be referred to as the 'natural ordering' of the jobs.

From assumption 1 the penalty function for job j becomes

$$p_j L_j = p_j \max\ (0; T_j - d_j) = p_j (T_j - t_j).$$

Since the objective function, $\sum_{j=1}^{N} p_j (T_j - t_j)$, is nondecreasing for T_j, a sequence produces a schedule in which all jobs start as early as possible for the given sequence. Therefore, if there exists an optimal schedule in which no job is split(Lemma 1), it is sufficient to examine at most N! different sequences in the search for an optimal solution.

Lemma 1: If $d_j = t_j$ for all j(assumption 1), there exists an optimal schedule in which no job is split between two or more machines.

Lemma 2: If M=1(i.e., if there is only one machine available), an optimal schedule is Q:(1, 2, $\cdots$, N). That is, the jobs are ranked in ascending value of the ratios t_j/p_j.

Corollary 1: For a schedule Q to be optimal, it is necessary for $Q_m: (m_1, m_2, \cdots, m_k)$ to be sequenced in the order of $r_{mj} \leq r_{mj+1}$, $j<k$, where k jobs are scheduled on machine m.

Corollary 2 : There exists an optimal schedule in which job 1 is scheduled first.

Lemma 3: For a schedule Q to be optimal, it is necessary that $T_{iv} \geq S_{jv}$ for every pair of machines $i$and j, where T_{iv} is the completion time of last job v on machine i and S_{jv} is the start time of last job v on machine j.

Theorem 1: If $t_i \leq t_j$ and $p_i \geq p_j$, then job i precedes job j in an optimal schedule.

Corollary 3: If $p_j = p_0$ for all j where p_0 is an constant, then there exists an optimal schedule in which jobs are assigned in their 'natural order' from machine 1 to machine M in rotation.

Theorem 2: An optimal schedule on two machines is s.t. job h precedes job k, if the following two conditions are satisfied:

(i) $r_h < r_{h+1}$ and $r_{k-1} < r_k$(note the strict inequality)

(ii) $\sum_{i=1}^{h-1} t_i \leq H - (1/2)\max_i t_i - \sum_{i=k}^{N} t_i$, where

$H = \sum_{i=1}^{N} t_i / M$.

Corollary 4: In the case of $M \geq 3$ machines, an optimal schedule is s.t. job h precedes job k if the following two conditions are satisfied:

(i) $r_h < r_{h+1}$ and $r_{k-1} < r_k$

(ii) $\sum_{i=1}^{h-1} t_i \leq H - \max_i t_i - \sum_{j=k}^{N} t_j$,

We have introduced important results on job orderings in an optimal schedule. In order to make these results clear, an illustration is given below.

Example 1: Suppose we are given jobs with following t_i and p_i for each job i. Assuming $M=2$, find some possible job ordering for an optimal schedule.

Data :

job	1	2	3	4	5	6	7
t_i	4	3	3	3	4	1	2
p_i	12	8	7	4	5	1	1

Notice that the jobs are already numbered by increasing t_i/p_i ratios. For this small size of problem, if we enumerate explicitly, there are 7!(i.e., 5040) sequences to be examined. We want to show how Corollary 2, Theorem 1 and 2 work to reduce the computational effort.

Corollary 2: shows job 1 precedes all other jobs in an optimal schedule,

Theorem 1: shows that job 1 precedes job 5 (because $t_1 \leq t_5$ and $p_1 \geq p_5$), job 2 precedes jobs 3, 4, and 5, job 3 precedes jobs 4 and 5, and job 6 precedes job 7.

Theorem 2: A more detailed explanation may be necessary for this theorem. Let us pick up job 2 and job 6 arbitrarily, and test whether the two conditions are satisfied.

(i) $r_2 = 3/8 < r_3 = 3/7$ and $r_5 = 4/5 < r_6 = 1$

(ii) $t_1 = 4 \leq 10 - 2 - 3 = 5$

Both two conditions are satisfied. Therefore, job 2 precedes job 6 in an optimal schedule. By the same procedure, we can also find that job 1 precedes jobs 5, 6, 7 and job 2 precedes job 7.

Combining the above informations on optimal ordering, we can find the following network-type precedence relationships among jobs.

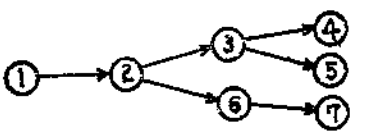


Figure 1: Precedence Relations

The above precedences reduce the number of sequences that must be considered to only 20 (a reduction of over 99%) sequences. We have investigated precedence relations among the jobs so that it is not necessary to search or an optimal schedule over all possible schedules, but only over a subset of the schedules. Further reduction of computational effort may result from the application of the branch-and-bound technique which will be described below.

3. Implicit Enumeration Procedure

We do not elaborate here on the principles of implicit enumeration by branch-and-bound methods The interested reader may wish to consult Agin [1], but briefly state two basic concepts of the method: one is that all feasible sequences must be accounted for either explicitly or implicitly, and the other is that the least number of sequences should be accounted for explicitly. The gist of the approach presented below is to try to limit the explicitly enumerated nodes by precedence relations and bounding arguments. Define a "partial schedule" to be a specification of the start times of a subset of the jobs which have been scheduled on the machines. Let a "partial permutation" denote a partial sequence (i.e., a permutation of some subset of the jobs 1, 2, $\cdots$, N). We define a partial schedule to be "eligible" if there does not exist a job which can be started earlier without changing the start times of some others in the partial schedule. Similarly, a partial permutation is "eligible", if job i precedes job j (from Corollary 2, Theorems 1 and 2), and j does not appear before i in the partial permutation. Schrage[15] shows that there is a one-to-one correspondence between an eligible partial schedule (e.p.s.) and an eligible partial permutation (e.p.p.). Also define an e.p.p. to be "redundant" if there exist two jobs i and j such that job i appears before job j in the e.p.p.; however, when jobs are scheduled as they appear in the permutation, then either $s_i > s_j$ or $s_j = s_j$ and $i > j$. Schrage observes that "redundant e.p.p. need not be considered in the enumeration process", which plays a role in eliminating partial permutations. We will take advantage of his statement to reduce the number of sequences which must be accounted for. However, the major roles in such reduction come from the bounding procedure by upper and lower bounds.

Upper Bound

The total cost of lateness for any complete schedule can be an upper bound. If we know a reasonably good upper bound on the optimal value in the midst of the tree generation process, it will be very useful because any e.p.p. whose lower bound is greater or equal to the upper bound can be discarded. For the tree generation we use backtracking instead of jumptracking. There fore, we get an upper bound on the optimal value as soon as we obtain a complete schedule with an associated total cost. It is quite possible that one obtains a better upper bound as one proceeds further in the tree generation. In this case, the latter bound replaces the former. This process for generating an upper bound is one advantage of the backtracking process, besides saving memory space in the computer.

Lower Bound

This is the major part of improvement as compared with the previous paper [6]. Suppose we want to evaluate the lower bound on the optimal value for an e.p.p., $P(k) = (J(1), J(2), \cdots, J(k))$ where $J(i)$ denotes the job in position i; $i = 1, 2, \cdots, k$.

$P(k)$ indicates an eligible partial permutation whose order is $J(1), J(2), \cdots J(k)$. The lower bound for $P(k)$ is obtained as follows.

$$\text{lower bound; } \mathrm{LB}(P(k)) = \sum_{j \in P(k)} p_j (T_J - d_j) + T_J \sum_{j \in (N)-P(k)} p_j + \max(0;\ C_{1,N-k}/M - P_{N-k}(M-1)/(2M)) \quad \cdots\cdots (1)$$

where (N) : set of all N jobs,

P_{N-k} : $\sum_{i \in (N)-P(k)} p_i t_i$

$C_{1,N-k}$: cost of processing $(N)-P(k)$ jobs in their natural order on a single machine,

T_J : (min T_i: i is the last job on each machine).

Proof: The first term in the right hand side of the above equation is the cost incurred in scheduling the k jobs in $P(k)$ and the last two terms arise from the consideration of the unscheduled jobs. The second term is self-evident, but the last term needs explanation. We derive the last term from Eastman, Even and Isaacs' result[4] on bounds for the optimal scheduling of N jobs on M processors. They prove that

$$C'_{M,N} - P_N/2 \geq (C'_{1,N} - P_N/2)/M \quad \cdots (2)$$

where P_N : $\sum_{i=1}^{N} p_i t_i$

$C'_{M,N}$: cost of processing N jobs on M machines where $d_j=0$ for all j,

$C'_{1,N}$: cost of processing N jobs in natural order on a single machine where $d_j=0$ for all j.

Their result is directly applicable to our model, since we assume $d_j=t_j$ for all j, that is, $T_j \geq d_j$ for all j. Therefore,

$$C'_{M,N} = C_{M,N} + P_N$$
$$C'_{1,N} = C_{1,N} + P_N$$

and inequality (2) implies that

$$C_{M,N} + P_N - P_N/2 \geq (C_{1,N} + P_N - P_N/2)/M$$
$$\longrightarrow C_{M,N} + P_N/2 \geq (C_{1,N} + P_N/2)/M$$
$$\longrightarrow C_{M,N} \geq C_{1,N}/M - P_N(M-1)/(2M).$$

Hence, the third term for the lower bound of $P(k)$ becomes

$$C_{1,N-k}/M - P_{N-k}(M-1)/(2M).$$

If M is very close to or equal to N, which are trivial cases, it might be possible to have

$$C_{1,N-k}/M < P_{N-k}(M-1)/(2M).$$

So, we write the third term as

$$\max(0;\ C_{1,N-k}/M - P_{N-k}(M-1)/(2M),$$

as shown in equation (1). Q.E.D.

The efficiency of a branch-and-bound algorithm depends on the branching rule to build a search tree. Since we use a particular form of branching procedure, we need to explain it before explicitly stating the algorithm.

Branching Procedure

Suppose there is a given $P(k)$. Define $A(P(k))$ to be the set of unscheduled jobs which are available to be scheduled in position $k+1$, appended to the right of $J(k)$ in $P(k)$. The branching procedure can best be explained by an example. Consider Figure 1. Obviously $P(1)$: $(J(1)=1)$ and $P(2)$: $(J(1)=1,\ J(2)=2)$. By definition of $A(P(k))$, $A(P(2))$ should be jobs 3 and 6. Notice that jobs 4, 5 and 7 cannot be a candidate for the next branch, since job 3 or 6 is not scheduled yet. Usually the job for the next branch is chosen by computing lower bound values for jobs 3 and 6, then selecting the job which has the smaller lower bound value. However, we do not adopt this usual branching rule. Instead, we use the rule "branch from the job which has the lowest job-number". Then, for the current network, $J(3)=3$, $P(3)$: $(1, 2, 3)$ and $A(P(3)) = (4, 5, 6)$. The reasoning is based on the following arguments.

(i) This rule is much simpler than the usual rule from the point of computational time, particularly in the case where there is

a large number of jobs in $A(P(k))$.

(ii) The fluctuation of job ordering for an optimal schedule is not expected to be far away from the natural ordering of jobs: Corollary 1 asserts this fact.

These two arguments turn out to be validated by the computational experiences. The number of iterations to reach the best-solution is relatively very low, as compared to the number of iterations to prove optimality. Consecutive generation of $A(P(k))$ can be done by the following scheme. Suppose we select job i from $A(P(k-1))$ and schedule it in position k. Then,

$$A(P(k)) = A(P(k-1)) - (i) + (j: j \in A(i) \text{ and } B(j) \subseteq P(k)),$$

where $A(i)$ = (set of immediately succeeding jobs of i), and

$B(j)$ = (set of immediately preceding jobs of j).

The Algorithm

A branch-and-bound algorithm for scheduling N jobs on M identical machines in parallel, where the objective is to minimize the total cost of lateness, is presented in the form of flow-chart.

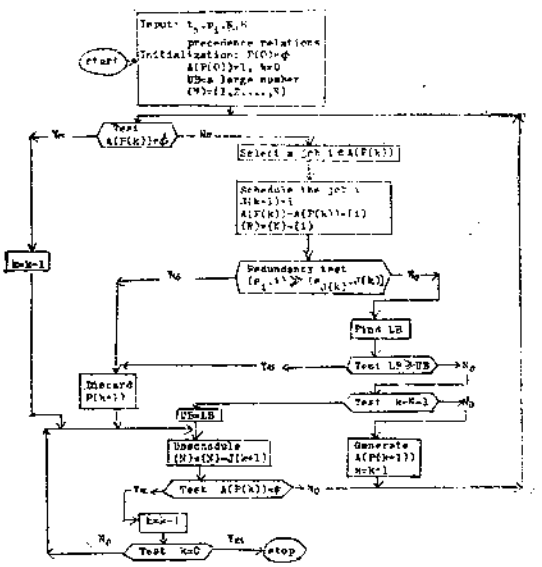


Figure 2: Flow chart
Legend: UB=upper bound
LB=lower bound
$\gg$=lexicographically greater than

The detailed step-by-step explanation of the flow chart may be found from the paper [6]. Note that the calculation of lower bound has been changed and the improved lower bound should be identified by equation (1). The effect of the change in lower bound will be discussed in the computational experiences. The following simple example illustrates how the branch-and-bound algorithm develops to find an optimal solution and prove its optimality.

Example 2: Find an optimal sequence which minimizes total cost of lateness when N=5, M=2, t_i=2, 4, 6, 5, 3, and p_i=2, 2, 4, 10, 9 for the jobs, respectively.

If we arrange the data in order of increasing $r_i = t_i/p_i$, the jobs are numbered as below.

job number	1	2	3	4	5
t_i	3	5	2	6	4
p_i	9	10	2	4	2

For this small size of problem, if we enumerate explicitly, there are 5! (i.e., 120) sequences or 600 nodes (iterations or partial permutations) in the search tree to be examined. Reductions of computation come from two sources; one from the precedence relations (Corollary 2, Theorems 1 & 2), and the other from the bounding procedure (mainly the lower bound).

Corollary 2: shows job 1 precedes all other jobs in an optimal schedule,

Theorem 1: shows that job 1 precedes jobs 4 and 5, job 2 precedes job 4, and job 3 precedes job 5 in an optimal schedule,

Theorem 2: shows jobs 1 and 2 precede job 5 in an optimal schedule.

Combing the above informations on optimal ordering of the jobs, the following network-type precedence relations may be found.

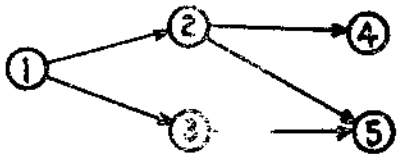
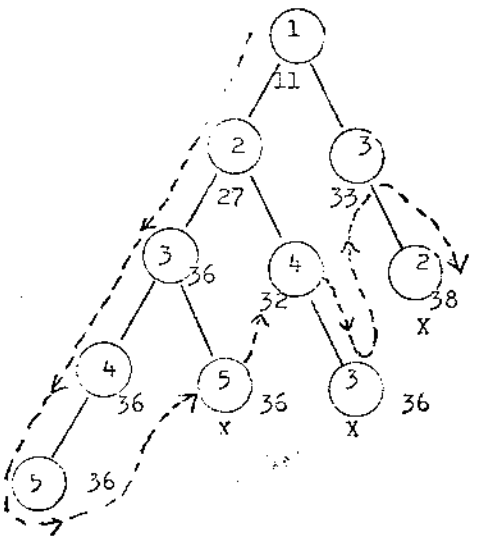
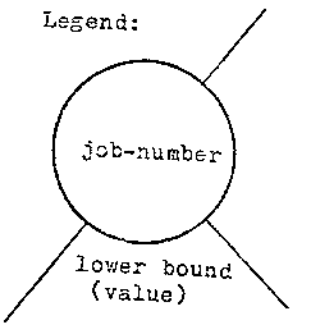


Figure 3: Precedence Relations

The above precedences reduce the number of permutations that must be accounted for to only 5 permutations (i.e., 1-2-3-4-5, 1-2-3-5-4, 1-2-4-3-5, 1-3-2-4-5 and 1-3-2-5-4), which have only 16 nodes (a reduction of over 95%). Further reduction results from the application of the lower bound or possibly from the redundancy test due to Schrage (15). Figure 4 shows the search tree which is created as the algorithm proceeds.

The example has a minimum total penalty cost of 36 and the optimal sequence is 1-2-3-4-5. We generate 10 out of the possible 16 nodes by using the bounding procedure.





X: indication of discard
Arrow: direction of the algorithm flow

Figure 4:Search Tree

4. Computational Experience

In order to compare the efficiency of the current branch-and-bound algorithm with the previous one, the exactly same problems appearing in [6] are tested. The program was used on an IBM 370/165 with PL/1 language for the compilation of the program. The computational results for the 15 problems are summarized in Table 1.

The computational experience is designed to investigate the increase in computational time as the number of jobs increases. The influence of the number of machines on computer time is assumed negligible based on the results of [3]. The solution time does not necessarily increase as the number of jobs increases, probably because the solution time is heavily dependent on the information structure of t_i and p_i for each problem. An extensive search for this dependency has not been attempted. However, a study of the results in Table 1 reveals the following:

(i) The computational time by the current algorithm is uniformly less than that by the previous algorithm. In particular, the reduction in time for the higher numbers of jobs

Problem Number	Number of Jobs	Number of Machines	Value of first solution	Value of best obtained	Computation time in seconds (cpu time) previous	current
1	5	2	28	26	0.4	0.3
2	6	2	60	60	0.3	0.2
3	7	3	28	26	0.4	0.4
4	8	2	131	129	0.5	0.4
5	9	3	140	140	0.4	0.4
6	10	2	718	714	6.1	5.4
7	11	3	109	109	1.2	1.0
8	12	2	489	489	0.9	0.8
9	13	3	232	230	3.6	2.7
10	25	3	952	950	20.7	18.7
11	35	4	1,078	1,076	8.4	7.2
12	45	4	4,052	4,052	57.1	52.3
13	55	8	2,841	2,835	82.5	62.7
14	65	8	3,465	3,456	97.8	68.0
15	75	8	2,510	2,502	193.9	120.5

Table 1. Computational Results

(n=55, 65, and 75) is noticeable.

(ii) The value of the first solution (i.e., the solution by the 'natural order') is very close to the optimal solution. Furthermore, five out of the 15 problems solved achieve the optimum on their first complete permutation. This implies that the branching procedure is very efficient.

(iii) Less than a second is required to solve the first five problems, less than a minute is need to solve the problems with 10 to 45 jobs, and about less than two minutes consumed for the problems with 55 to to 75 jobs, which indicate that such moderately large-sized problems can still be solved within quite short periods of computer time.

5. Summary

An improved branch-and-bound algorithm was discussed for the sequencing problem when all jobs are available to process at time zero and are independent. The computer times required to solve the problem are encouragingly short, which demonstrates that the algorithm is rather efficient. The following factors have contributed to make the algorithm efficient.

(i) The reduction of partial permutations by Corollary 2, Theorems 1 and 2 is significant. These results, in fact, bring computationally infeasible sequencing problems to within the realm of computability.

(ii) The improved lower bound is very sharp to reduce the computational effort effectively.

(iii) The unusual branching procedure based on the job-numbering schemes is very efficient.

(iv) The back-tracking method adopted in the search tree is efficient. Note that many branch-and-bound methods use the jump-tracking method.

Last of all, we would like to mention that the program listing is available from the author for any interested person.

References

[1] Agin, N. "Optimum Seeking with Branch and Bound," *Management Science*, 13, 4, (December 1966), pp. B-176-185.

[2] Arthanari, T.S. and K.G. Ramamurthy, "A Branch and Bound Algorithm for Sequencing n-Jobs on m-Parallel Processors," *Opsearch*, 7(1971), pp. 147—156

[3] Bhadury, B., "An Experimental Investigattion of a Model for Sequencing n-Jobs on m Machines in Parallel," Unpublished thesis, North Carolina State Univ. at Raleigh(1970).

[4] Eastman, S., S. Even, and I. Isaacs, "Bounds for the Optimal Scheduling of n Jobs on m Processors," *Management Science*, 11, 2, (November 1964), pp. 268—279.

[5] Elmaghraby, S., "The Sequencing of n Jobs on m Parallel Processors with Extensions to the Scarce Resource Problem of Activity Networks," Proceedings of the Inaugural Conference of the Scientific Computation Center and the Institute of Statistical Studies and Research, Cairo, Egypt,(December 1968)

[6] Elmaghraby, S., and S. Park, "Scheduling Jobs on a Number of Identical Machines" *AIIE Transactions*, 6, 1, (March 1974), pp. 1—13.

[7] Mason, A. and C. Moodie. "A Branch and Bound Algorithm for Minimizing Cost in Project Scheduling," Research Memo. No. 70-2, School of Engineering, Purdue Univ., Lafayette, Ind. (1970).

[8] McNaughton, R. "Scheduling with Deadlines and Loss Functions," *Management Science*, 6, 1 (September 1959), pp. 1—12.

[9] Nabeshima, I. "On the Bound of Makespan and its Application on m Machine Scheduling Problem," *Journal of Operations Research Society of Japan*. 9,3 & 4 (July 1967), pp. 98—136.

[10] Nabeshima, I. "Some Extensions of the m Machine Scheduling Problem," *Journal of Operations Research Society of Japan*, 10, 1 & 2 (Oct. 1967), pp. 1—17.

[11] Park, S. "A Branch-an d-Bound Method for the Sequencing Problems of n Jobs on m Identical Machines in Parallel," Unpublished thesis, North Carolina State Univ. Raleigh, N.C. (1972).

[12] Pritsker, A.,L. Watters, and P.Wolfe, "Multiproject Scheduling with Limited Resources: A Zero-one Programming Approach," *Management Science*, 16.1 (Sept. 1969), pp. 93—108.

[13] Root, J., "Scheduling with Deadlines and Loss Functions on k Parallel Machines," *Management Science*, 11, 3, (Jan. 1965), pp. 460—475.

[14] Rothkopf, M., "Scheduling Independent Tasks on Parallel Processors," *Management Science* 11, 3 (Jan. 1966), pp.437~447.

[15] Schrage, L., "Solving Resource Constrained Network Problems by Implicit Enumeration Non-preemptive Case," *Operations Research*, 18, 2, (March 1970), pp.263—278.