*Proceedings of the 1997 Winter Simulation Conference*
*ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson*

# EFFICIENT SIMULATION/OPTIMIZATION OF DISPATCHING PRIORITY WITH "FAKE" PROCESSING TIME

Susumu Morito
Keun Hyung Lee

Department of Industrial and
Management Systems Engineering
Waseda University
3-4-1 Ohkubo, Shinju-ku, Tokyo 169, JAPAN

## ABSTRACT

An efficient simulation/optimization approach is presented for finding a dynamic dispatching priority in a static job shop environment under the presence of multiple identical jobs. The key ingredients of the proposed approach are (1) an efficient processing-time-based dispatching rule, (2) simulation of a job shop, and (3) a mechanism to "fake" (or modify) job processing times based on the information of job slack obtained from simulation. The paper presents an overall approach to fake processing times, and also identifies alternative strategies for algorithm design.

Experimental results of the proposed approach yield a dispatching priority with more than 10% better due date performance than the priority generated by a good processing-time-based dispatching rule in roughly 20 simulation runs.

## 1 INTRODUCTION

Simulation has been regarded as a powerful tool for short-term look-ahead from early days. Nowadays, with fast PCs and workstations and with powerful simulation software, simulation-assisted look-ahead is not a dream but a reality.

Dispatching rules have been used widely in scheduling job shops. Many dispatching rules have been proposed, analyzed, and in fact used to schedule jobs. However, there has been little effort in combining simulation with dispatching. The only exception, to the best of the authors' knowledge, is Vepsalainen and Morton (1988).

This paper presents a simulation/optimization approach to generate a good dispatching priority using a simple processing-time-based dispatching rule as a basis. The paper is organized as follows. The following section describes job shops with multiple identical jobs and its control via dispatching. The way we use two terms, dispatching rule and dispatching priority, is clarified, and the problem addressed in this paper is defined. Section 3 describes related research. Section 4 presents the proposed approach. The descriptions start from the overall approach, then move on to a mechanism to fake processing time and a local search, and end up with the algorithm statements. Section 5 gives descriptions of the experiments and their results. The last section gives conclusions.

## 2 JOB SHOP WITH MULTIPLE IDENTICAL JOBS AND ITS CONTROL VIA DISPATCHING

### 2.1 Job Shop with Multiple Identical Jobs

We consider a "static" job shop scheduling problem, in which a set of jobs to be scheduled are known and available at time 0. In most static job shop scheduling problems, it has been generally assumed that each job is unique. That is, each job has or can have unique job characteristics such as routing and processing time.

In reality, however, production requests often arrive in the form of production "lots"(other possible names include "batches" or "orders") each of which calls for a specified number of parts of the same type. For example, production order $XYZ$ requests 25 pieces of part type $A$ to be completed by 12 noon, April 1, 1997, and production of part type $A$ consists of a set of "operations", say, $A_1$,$A_2$,$A_3$ which must be processed in this order. The required number of parts for a lot is called a "lot size".

When production requests come in the form of production lots and each lot consists of as many jobs of the same type as the lot size, many identical jobs would flow through a shop. Under the condition where machine setups due to change of job types incur major cost, one tends to produce the entire lot all at once, and if so, each lot can be regarded as a single job and the job shop can be viewed as one with unique jobs.

However, in many contemporary automated job shops such as FMS's, each lot need not or may not be regarded as a single job because of the following reasons:

1) Differences of routing, lot sizes, and processing times of different part types all contribute to "dis-

tributed" processing of jobs that constitute a lot.

2) A limited number of pallets and fixtures makes it impossible to process jobs that constitute a lot all at once.

3) Some form of material handling system which is often automated, is used and required to transport works placed on pallets.

4) Hardware is designed to cope with simultaneous production of many distinct works, and the associated control mechanism, i.e., software does not try to group all jobs constituting a lot as one big batch.

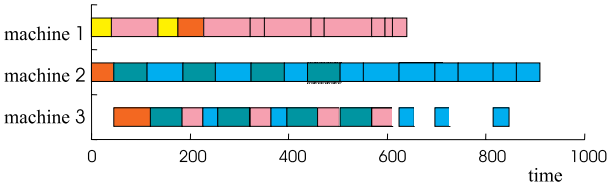


Figure 1: Job Shop with Multiple Identical Jobs

When the multiple identical jobs belonging to a particular lot cannot or need not be processed all at once, the associated Gantt chart may look like Figure 1. The shop like Figure 1 creates difficulties when one wants to estimate the production lead time of a particular lot, which is the time between the start and the end of the entire lot. If a lot calls for, say, 25 pieces of part type $A$ with three operations, $A_1$,$A_2$,$A_3$ to be processed in this order, time between the start of operation $A_1$ of the first job and the end of A3 of the last (i.e., 25th) job, constitutes the production lead time. Since, for example, the second operation $A_2$ of the first job can be started after the completion of $A_1$ of the first job, and need not wait for the completion of $A_1$ of the entire lot, production of a particular lot "overlaps" time-wise. Furthermore, "interference" with jobs belonging to other lots generally creates waiting. Such interference together with overlapping production makes it difficult to estimate production lead time of a lot. A lack of knowledge for the production lead time would make scheduling difficult in a job shop with multiple identical jobs.

### 2.2 Dispatching Rule and Dispatching Priority

Throughout this paper, we distinguish "dispatching rules" from "dispatching priorities". Here, a dispatching priority is defined to be a linear ordering of all relevant operations, based on which jobs are dispatched on all machines. If, for example, operations to be dispatched consist of $A_1$,$A_2$,$A_3$; $B_1$,$B_2$; $C_1$,$C_2$,$C_3$ , any one of the linear orderings of the 8 operations, say,

$C_2 \rightarrow A_3 \rightarrow B_1 \rightarrow B_2 \rightarrow C_1 \rightarrow A_2 \rightarrow C_3 \rightarrow A_1$

gives a dispatching priority. We assume that a single dispatching priority is shared on all machines.

In a job shop with multiple identical jobs, two or more distinct operations of the same part type, say $C_1$ and $C_3$, may compete for the same machine, and it is not enough to assign priority to jobs, and thus becomes necessary to assign priority to operations.

A dispatching rule, on the other hand, is defined to be a well-defined mechanism to determine a dispatching priority based on such information as job properties(lot size, lot due date, number of operations and the associated processing time, routing, etc.), current shop status, as well as current time. As in Figure 2, a dispatching rule can be regarded as a mechanism which maps input information to a linear ordering of operations.

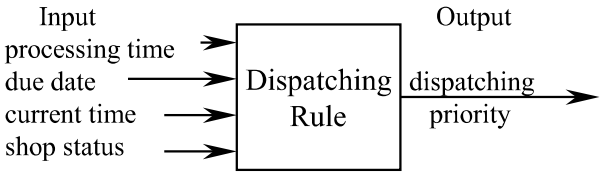


Figure 2: Dispatching Rule and Dispatching Priority

### 2.3 Problem Statement

We now define a "dispatching priority problem", which is a short-term scheduling problem. Note that we focus our attention on the "dynamic" dispatching priority, i.e., the priority ordering changes dynamically over time reflecting changes of production status. Here, we take the total tardiness as the performance measure, even though other measures could have been used.

**Objective**: Minimize total tardiness

**Conditions**: Each part type is given a lot size and a due date by which the production of the entire lot is hoped to be completed. All processing information such as the associated operations together with their sequence and time requirements are assumed to be known and deterministic. Detailed descriptions of the simulation model will be given in 5.1.

**What is to be decided**: Dispatching priority of parts and operations

Finally, we note that the particular type of static scheduling problems presented is important in automated job shops such as FMS's with multiple identical jobs because:

1) System control software often requires a system user to specify dispatching priority, or at least ask the user to select a dispatching rule to be used.

2) A set of lots to be processed concurrently are usually known (i.e., have been decided) in advance. A decision to select which lots to be put into the

system is often referred to as part type selection, and we assume that this decision has been made already.

## 3- RELATED RESEARCH

### 3.1- Use of Simulation in Dispatching

An iterative use of simulation in the determination of dispatching priority was probably first proposed by Vepsalainen and Morton(1988).

### 3.2- Use of "Modified Information" in Dispatching

There are some attempts in the past to use "fake" data to generate a dispatching priority. Probably the most well-known and well-performed one would be an extremely simple scheme called Modified Due Date (MDD) of Baker and Kanet (1983).

### 3.3- Optimization of Dispatching Priority

Morito and Lee (1994) tried to explore the best dispatching priority with the least total tardiness among all possible priority ordering. They noted the fact that any dispatching priority with good performance is acceptable even if one does not know a rule (i.e., a mechanism) which relates the priority ordering to the jobs' properties and other factors. A combination of simulation with simulated annealing is employed to explore a near-optimal dispatching priority. They assumed a static job shop environment (i.e., all jobs are available at time 0, and no new jobs arrive thereafter) and the static dispatching priority (i.e., priority does not change over time).

Their study showed that the best dispatching priority found by their simulation/optimization approach produced slightly more than 10% better due-date performance than best dispatching rules they compared. The gain was obtained, however, after thousands of iterations of their simulation/optimization approach. Despite their attempt, they failed to come up with a new rule that yields the best dispatching priority obtained by the simulation/optimization approach.

The results indicate that there is a potential for better dispatching rules, but it is not easy to find such ones. Optimization of dispatching priority is interesting, but its computational burden generally appears to be too heavy. The question then would be: Are there ways to come up with a good dispatching priority without "wasting" a huge amount of computation? This paper addresses the above question, and presents a simple and effective simulation-based mechanism to fill the gap.

## 4- A SIMULATION-ASSISTED SCHEME FOR OBTAINING A GOOD DISPATCHING PRIORITY

### 4.1- Basic Design Philosophy

We first describe our basic design philosophy:

1) *Use a well-performed processing-time-based dispatching rule to generate an initial dispatching priority.*

Our past studies indicate that the processing-time-based dispatching rules generally produce an excellent due-date performance without sacrificing shop utilization (Lee, Morito, and Imaizumi 1996). Obviously, these rules are simple and take little time. Therefore, it is natural to use such a rule to generate an initial dispatching priority.

2) *Use simulation to obtain estimates for job completion time(or equivalently, job slack or tardiness).*

Provided that a simulation model of a job shop is available, job completion times can be estimated by simulating the shop assuming the given dispatching priority. This allows us to estimate slack or tardiness of jobs.

3) *Intelligently "fake" processing time of jobs and apply again the processing-time-based dispatching rule to generate a dispatching priority with better due-date performance.*

If simulation results indicate that a particular job, say job $A$, has a lot of slack with some other jobs being tardy, one naturally wants to lower the dispatching priority of job $A$ so that tardiness of other jobs can be reduced. Assuming that we use a simple processing-time-based dispatching rule, this might be achieved by "increasing" the processing time of job $A$ for the purpose of sorting in priority generation. In other words, we "fake" the processing time of job $A$. Thus, we fake processing times of some jobs systematically based on some mechanism in the hope that due-date performance is improved, and re-apply the selected processing-time-based dispatching rule to generate a new and hopefully "better" dispatching priority.

Based on the general philosophy stated above, we now present the outline of a simulation-assisted scheme to find a reasonable dispatching priority. First, we present an overview of the entire approach, followed by brief descriptions of its key ingredients, which consist of 1) a precise mechanism of processing time fake, and 2) an overall design of an iterative method to explore an improved dispatching priority.

### 4.2- Overall Approach (Figure 3)

Figure 3 schematically shows the overall approach. Entries in the boxes indicate some sort of mechanisms whose input and output are indicated in italics.

Given job data, we first apply a specified processing-time-based dispatching rule to generate a specific dispatching priority. We note that when the processing-time-based dispatching rule is fixed, a given set of processing times lead to a unique dispatching priority (unless there exist ties). Let us call the resultant priority the "base" dispatching priority.

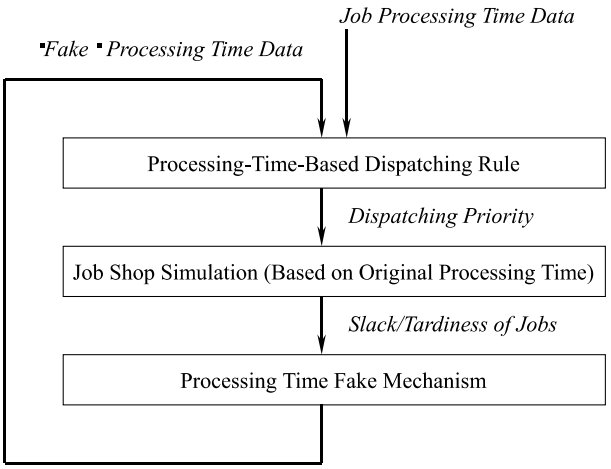


Figure 3: Overall Approach

Under the current technology, the only approach possible to estimate shop performance of the given base dispatching priority appears to be simulation. Thus, we run simulation to obtain its shop performance. In particular, we obtain slack/tardiness information of jobs from simulation results. Note that the simulation is run based on the original processing time data, and not based on the fake data. Given the slack/tardiness information, we perform processing time fake. Given the base dispatching priority, we try several alternative fake data sets. This is because a particular fake may not lead to improved dispatching priority.

Once a set of fake processing times are given, we again apply the dispatching rule to obtain the associated dispatching priority, which will be simulated to obtain slack/tardiness information. There are several possible ways to organize the search, such as a local search, a branch and bound, and a backtrack. Below, we develop an algorithm based on a local search, in which different sets of fake processing times constitute a "neighborhood" of a given base dispatching priority.

In a local search with "best improvement strategy", the algorithm continues to find the dispatching priority among a neighborhood which yields the least tardiness. If no improvement is achieved as compared with the base dispatching priority, we stop. Otherwise, replace the base dispatching priority with the newly-found improved dispatching priority and essentially the same procedures repeat until termination.

### 4.3- Design of "Fake" Mechanism

Our basic idea is simply to fake job processing times with positive slack to make it longer. A variety of mechanisms can be considered to achieve this goal. Therefore, determination of a precise mechanism to fake processing time of jobs requires us to study carefully several alternative design strategies. Below, we list several view points for such mechanism design.

1) *Do we apply only "positive" fake (increasing processing time) or both positive and "negative" fake (decreasing processing time)?*

2) *Do we relate the magnitude of slack/tardiness to the degree of processing time modification?*

3) *How do we decide the absolute magnitude of processing time modification?*

4) *Do we include randomness in the fake mechanism?*

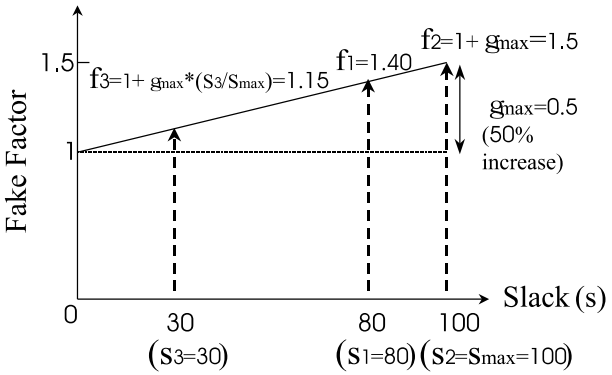


Figure 4: Processing Time Fake Pattern

In this paper, we employ the deterministic mechanism which considers only "positive" fake to increase processing time of jobs with positive slack. An intuitive justification for not applying fake to tardy jobs is the well-known fact that the SPT (Shortest Processing Time first) rule is optimal for 1-machine mimimum flow time scheduling, and is expected to produce good due date performance in a more general shop environment. Thus we do not feel it necessary to fake processing times of tardy jobs.

A linear relationship between the relative fake strength and the magnitude of slack is considered which applies stronger modifications for jobs with bigger slack. Figure 4 shows a basic "fake pattern", in which the horizontal axis corresponds to the magnitude of job slack, and the vertical axis, the magnitude of processing time fake (called the fake factor). If, for example, the fake factor $f_3$ for job 3 is 1.15, then the processing time of job 3 will be increased by 15%. The maximum value of the fake factor $1+g_{\max}$ (=1.5 in Figure 4) given to the job with the biggest slack will be specified as described below.

### 4.4- An Iterative Method Based on Local Search

Faking processing times based on such a mechanism discussed above and applying a processing-time-based dispatching rule does not guarantee a generation of a better dispatching priority. That is, we must be prepared that the idea fails for a particular fake. If a fake fails to produce a better priority, we look for other fakes. We accomplish this by a form of a local search.

#### 4.4.1- Neighborhood

In designing a local search, selection of a neighborhood is essential. It is not at all obvious what kind of neighborhood is appropriate here, but we try to generate different fake patterns given a "base" dispatching priority by adjusting "strength" of the fake. We accomplish this by changing the maximum value of the fake factor. As a stronger fake tends to distort the original "problem structure", the algorithm tested here adopts the strategy to start from weaker fakes (i.e., less modification) and move towards stronger fakes (i.e., more modification).- For example, one neighbor is defined by $g_{\max} = 0.5$, and another defined by $g_{\max}$ =1.0(i.e., a maximum of 100% processing time increase), and so forth.

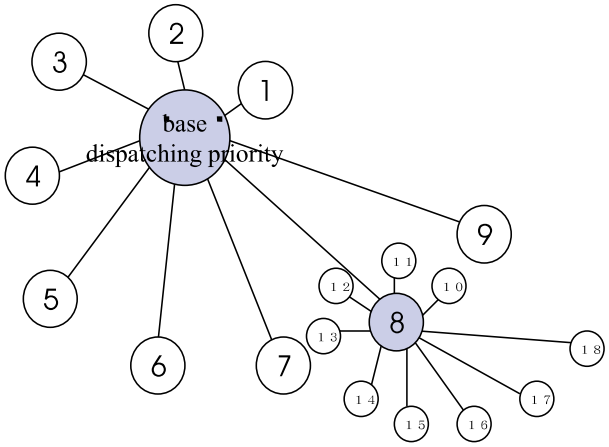


Figure 5: Neighborhood

Figure 5 shows the image of the local search. The figure assumes the best improvement strategy to select the best among the neighborhood which consists of 9 distinct fake data sets. The relative distance between nodes schematically implies the strength of the fake. Namely, node 1 (node 9) corresponds to a fake data set of the least (largest) strength in the neighborhood of the base dispatching priority. Figure 5 assumes that the fake pattern of node 8 produced the best improvement among the neighborhood, and thus the associated dispatching priority becomes a new base priority with the new neighborhood constituting nodes 10 through 18. The process continues unless there is no improvement in the neighborhood.

If the "first improvement strategy" is adopted, we switch the base dispatching priority as soon as an improved dispatching priority is found. If no imrovement is achieved within the specified upper limit, the algorithm stops. Some intermediate strategy between best improvement and first improvement could be devised, but will not be considered in this paper.

### 4.5- Prototype Algorithm

A prototype algorithm is now stated below:

**Step 0** (*Selection of the basic dispatching rule, initial generation of a dispatching priority, and initial base simulation*) Select a processing-time-based dispatching rule used as a building block. Apply the selected rule on a given set of job data, and produce an initial dispatching priority which is now called as the initial "base" dispatching priority. Perform a simulation run using original processing time information to evaluate the effects of the initial base dispatching priority. Initialize the "outer" iteration counter $k$ to 1. Set $O_{\mathrm{ij}}$, the measure for lot $i$, operation $j$, used to determine priority to conform with the selected processing-time-based dispatching rule (say, ODSMT as recommended by Lee, Morito, and Imaizumi 1996).

**Step 1** (*"Neighborhood" search with simulation*)
Repeat the neighborhood search, **Step 1h**, for $h = 1, ..., h_{\max}$, where $h_{\max}$ is the size of the neighborhood which is a predetermined parameter. Here, we assume that $s_{\mathrm{i}}$ denotes slack for lot $i$ as obtained from simulation with the base dispatching priority. That is, $s_{\mathrm{i}} = max(0, d_{\mathrm{i}} - c_{\mathrm{i}})$ where $d_{\mathrm{i}}$ denotes the given due date of lot $i$, and $c_{\mathrm{i}}$ the completion time of lot $i$ as obtained from the simulation of the base dispatching priority. Note that values of $s_{\mathrm{i}}$ do not change throughout **Step 1h** below, as the base dispatching priority may be updated only in **Step 2**.

**Step 1h** Determine the maximum incremental fake factor $g^{\mathrm{h}}_{\max}$ for $h$-th neighbor:

$$g^{\mathrm{h}}_{\max} = G(g^{\mathrm{h-1}}_{\max})$$

and calculate the perturbation factor (relative to the base case) $f_{\mathrm{i}}$ of lot $i$ via

$$f_{\mathrm{i}} = \begin{cases} 1 + g^{\mathrm{h}}_{\max}\,(s_{\mathrm{i}}/max_{\mathrm{i}}(s_{\mathrm{i}} \mid s_{\mathrm{i}} > 0)) & \text{if } s_{\mathrm{i}} > 0 \\ 1 & \text{otherwise.} \end{cases}$$

Determine the dispatching priority based on new measures $N_{\mathrm{ij}}$ where $N_{\mathrm{ij}} = O_{\mathrm{ij}} \cdot f_{\mathrm{i}}$.

Perform then a simulation run based on the resultant dispatching priority but with the original processing time. If the resultant total tardiness is less

than that of the current best dispatching priority, replace the current best dispatching priority and also save the associated values of $f_i$ as $f_i^*$.

**Step 2** (*Termination check and updating the base dispatching priority*)

If there is no improvement in the neighborhood (i.e., during **Step 1h**, $h = 1, ..., h_{max}$), stop. Use the current base dispatching priority. Otherwise, the current best priority has been updated during the previous sub-iterations, and renew the "base" dispatching priority to this newly-obtained current best priority. Increment the iteration counter $k$ by 1. Replace the measure $O_{ij}$ by $O_{ij} \cdot f_i^*$ (for all $i$ and $j$), i.e., $O_{ij} \leftarrow O_{ij} \cdot f_i^*$ (for all $i$ and $j$) where $O_{ij} \cdot f_i^*$ is the measure used to derive the current best dispatching priority, and return to **Step 1**.

The above algorithm is stated assuming the best inprovement strategy, but can be modified easily to other strategies. Other termination criteria, such as "forced termination" based on iteration counts and/or the number of simulation runs, can be accomodated easily.

### 4.6 Detailed Settings

As a base dispatching rule, we employ ODSMT, which gives the highest priority to an operation with the least value of $(l_i p_i \cdot l_i p_{ij})$, where $p_i$ is the total processing time of the part type of lot $i$, $p_{ij}$ the processing time of operation $j$ of part type of lot $i$, and $l_i$ the lot size of lot $i$. Rule ODSMT was the best dynamic dispatching rule tested in Lee and Morito(1996) with regard to the tardiness measure.

In the experiments to be described below, we use the following form of the function and parameter:

$$g_{max}^h = G(g_{max}^{h-1}) = g_{max}^{h-1} + a.$$

Here $a$ is a fixed constant, and the value of 0.9 is used in our experiments. Note that this is equivalent to say that $g_{max}^h$ is increased by 0.9 each time we strengthen the fake. The choice of $a$=0.9 is based on the results of preliminary experiments. The initial value $g_{max}^0$ is assumed to be 0.

## 5 EXPERIMENTS AND RESULTS

### 5.1 Model Assumptions

The model described below is a simplified version of a complex and detailed simulation model developed for a commercial module-type medium- to small-scale FMS. We envision a system which consists of machining centers, which are interconnected to load/unload stations (LUL) and pallet stocker by a stacker crane, even though our model here do not explicitly include LUL's, stocker, and a stacker crane, under the assumption that they are not constraining resources. Other important elements of the system considered include pallets (including fixtures), and finite local machine input buffer. We list basic assumptions of the model below:

1) Jobs to be processed correspond to a finite number of distinct part types. For each part type, a predetermined production quantity is known which we call a "lot size". Processing of a specific part type consists of several "operations", where each operation can be performed without interruption on a single machine. A given series of operations for a particular part must be performed in the prescribed order. We assume a one-to-one correspondence between part types and lots.

2) A work is placed on a pallet with operation-dependent fixtures. The number of pallets available for a particular operation is assumed to be a fixed constant, which is often a lower single digit number.

3) A pallet and the associated work(s) must be returned to LUL between any two operations. At LUL, a work is removed from the pallet and the next work requiring the same operation, if available, is loaded on this pallet. Subsequently, the removed work will be loaded on the pallet for the "next" operation, if one exists.

4) What we call an "operation" generally consists of many "sub-operations" and the number of tools required for an operation often reaches as high as 30 - 50. All these sub-operations will be performed automatically based on the prescribed programs of NC (numerically controlled) machines, and the time required for tool changes is practically negligible with the help of automatic tool changer (ATC). We assume that tool assignments are already given and the machine(s) which can process a particular operation is known, and thus we pay no attention to sub-operations. The machine(s) that can process a particular operation is (are) sometimes referred to as the "candidate machine(s)" for the operation.

5) Raw materials for the first operation of all parts are assumed to be available at time 0.

6) Each machine has a finite input buffer of a specified size. Whenever an empty space becomes available in a particular buffer, the system tries to fill the space. Therefore, the system works in a "pull" mode in a sense.

7) Any movement of pallets requires an "infinite-speed" stacker crane whose control is determined by the user-specified dispatching priority. That is, a dispatching priority as in (1) is scanned from the top (i.e., the operation with the highest priority) to see if there exists a (transportable) work corresponding to the operation of interest. If there is one, dispatch

the stacker crane to move the work. Since we assume an infinite-speed stacker crane, the work will be brought to the buffer immediately. If one cannot find the work, look for a work with the next highest priority, and so forth. We assume that there exists one static dispatching priority which is applied to any requests for transportation. In other words, the dispatching priority does not change over time, nor depends on machines.

8) No preemption is allowed during processing.

9) Processing time of operations, the number of pallets for a specific operation, due dates given to each lot of a part type, are all known and deterministic. Work transportation time and load/unload time are negligibly small, and they are assumed to be 0.

10) No equipment (machines, LUL's, a stacker crane, tools) failure is considered.

### 5.2- Experimental Conditions

The results presented below are based on a simple 10-machine job-shop-type FMS model. We assume that each lot calls for distinct part type, and the number of lots (which is same as the number of distinct jobs) is 20. Lot sizes are generated from uniform distributions in [1-5]. The number of operations per job is also generated randomly from uniform distributions in [1-5]. Processing time of individual operations follows a uniform integer distribution between 15 and 95 (min.). Only a single pallet is assumed to be available for each operation, and also the size of local machine input buffer is 1 for each machine. The number of machines capable of processing an operation is assumed 1. Finally, due dates are determined randomly based on the method with two parameters, namely, the tightness factor T and the range factor D, as in Morito and Lee(1994).

### 5.3- Experimental Results

#### 5.3.1- Overall Performance

Table 1 shows the performance of the proposed approach. Results shown reflect the average of 10 randomly generated problems. Two variations of the local search are tried, that is, the first improvement and the best improvement among neighborhood. The approach produced roughly 10% improvement over the initial dispatching priority of ODSMT with only 3 iterations and approximately 30 simulation runs (under the best improvement). Here, the number of iterations corresponds to the number of distinct base dispatching rules until a local optimum is reached.

Table 1: Performance of the Proposed Scheme

| | best imprv. | first imprv. |
|---|---|---|
| no. of iterations | 3.4 | 2.85 |
| no. of simulation runs | 34 | 18.58 |
| final improvement(%) | 10.20 | 9.53 |

#### 5.3.2- Process of Improvement

Figure 6 shows how improvement could be achieved within 52 simulation runs starting from the initial base dispatching priority obtained by ODSMT. The horizontal axis of the figure show the average % improvement achieved by the neighborhood search within the designated number of fake patterns. The results reflect the average of 100 distinct problems.

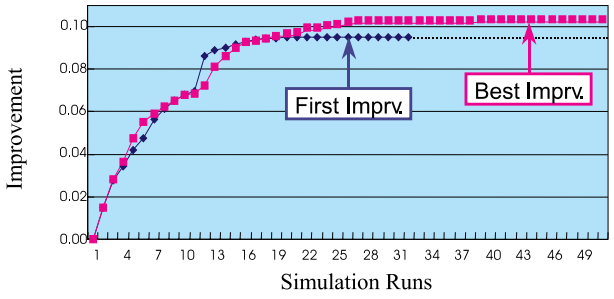


Figure 6: The Number of Different Fake Patterns Tested and Improvement Achieved

In the case of best improvement strategy, 10 different sets of fakes with varying strengths are generated (starting from the weaker ones to the stronger ones) and evaluated, and the plots show the best results up to the designated run. In the case of first improvement strategy, the base dispatching priority is updated as soon as an improved priority is found.

We can observe that 10% improvement can be achieved after 20 different fake patterns with different strengths. Figure 6 also shows that most of the gain is achieved with 20 iterations. This is understandable because the stronger fake tends to mess up the problem structure. The performance does not differ much depending on the best or first improvement strategy, but the best improvement gives slightly better results.

## 6- CONCLUDING REMARKS

We presented an efficient simulation/optimization approach for finding a dynamic dispatching priority in a static job shop environment under the presence of multiple identical jobs. The basic ingredients of the

proposed approach are an efficient processing-time-based dispatching rule, simulation of a job shop, and a mechanism to fake job processing times based on the information of job slack. Experimental results of the proposed approach are presented which show that an excellent dispatching priority can be obtained with a few iterations of the algorithms and also with a few simulation runs, without spending a huge amount of computer time.

Main contributions of the paper are in order:

1) An overall approach to fake (i.e., to modify) processing times to obtain better dispatching priorities is presented, and alternative strategies for algorithm design are identified.

2) The proposed approach generated a dispatching priority more than 10% better than the priority generated by a good processing-time-based dispatching rule. The margin was as big as 15%, and such good results were obtained within 20 simulation runs. Moreover, much of the gain was achieved during the first 5 simulation runs or so.

## ACKNOWLEDGMENTS

This research is supported by Waseda University Grant for Special Research Projects 96A-115, 96B-031,97A-135 and Ministry of Education Grant for Scientific Research (C)(2) 08680469.

## REFERENCES

Baker, K. R., and J. J. Kanet. 1983. Job shop scheduling with modified due dates. *Journal of Operations Management* 4:11-22.

Lee, K. H., S. Morito, and J. Imaizumi. 1996. Experimental evaluation of processing-time-based dispatching rules in a job shop with multiple identical jobs. *Journal of Japan Industrial Management Association (in Japanese)* 47:282-291.

Morito, S., and K. H. Lee. 1994. An application of simulated annealing for an FMS dispatching priority problem. In *Proceedings of New Directions in Simulation for Manufacturing and Communications*, ed. S. Morito, H. Sakasegawa, K. Yoneda, M. Fushimi, and K. Nakano, 197-203. Operations Research Society of Japan, Tokyo.

Vepsalainen, A. P. J., and T. E. Morton. 1988. Improving local priority rules with global lead-time estimates. *Journal of Manufacturing and Operations Management* 1:102-118.

## AUTHOR BIOGRAPHIES

**SUSUMU MORITO** is Professor of Operations Research at Department of Industrial and Management Systems Engineering, School of Science and Engineering, Waseda University. He received his B.Engr. and M.Engr. from Waseda, and also M.S. and Ph.D. from Department of Operations Research, Case Western Reserve University, Cleveland, Ohio. Professor Morito taught at Case Western Reserve and also at the University of Tsukuba prior to joining Waseda. His main area of research is operations research, with particular emphasis on applications of discrete-event simulation in manufacturing. He serves as the co-president of the special interest group on simulation at the Operations Research Society of Japan.

**KEUN HYUNG LEE** is a doctoral student at Department of Industrial and Management Systems Engineering, Waseda University. He holds Master of Management from Yokohama National University.