Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 424370, 7 pages
http://dx.doi.org/10.1155/2015/424370

Research Article

Heuristics for the Buffer Allocation Problem with Collision Probability Using Computer Simulation

Eishi Chiba

Department of Industrial and Systems Engineering, Hosei University, 3-7-2 Kajino-cho, Koganei-shi 184-8584, Japan

Correspondence should be addressed to Eishi Chiba; e-chiba@hosei.ac.jp

Received 13 January 2015; Accepted 18 May 2015

Academic Editor: Yannis Dimakopoulos

Copyright © 2015 Eishi Chiba. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The standard manufacturing system for Flat Panel Displays (FPDs) consists of a number of pieces of equipment in series. Each piece of equipment usually has a number of buffers to prevent collision between glass substrates. However, in reality, very few of these buffers seem to be used. This means that redundant buffers exist. In order to reduce cost and space necessary for manufacturing, the number of buffers should be minimized with consideration of possible collisions. In this paper, we focus on an in-line system in which each piece of equipment can have any number of buffers. In this in-line system, we present a computer simulation method for the computation of the probability of a collision occurring. Based on this method, we try to find a buffer allocation that achieves the smallest total number of buffers under an arbitrarily specified collision probability. We also implement our proposed method and present some computational results.

1. Introduction

Reflecting the increasing demand for Flat Panel Displays (FPDs) such as LCDs and plasma display panels, more effective methods for their manufacture are required. Production rate is continually improving with technological advances such as the rapid enlargement of glass substrates and the miniaturization of patterns. Accordingly, production lines have to be modified to accommodate such advances, and new optimization problems to be solved continue to arise. One introduction in the production line of FPDs is an advanced system called *Crystal Flow* [1]. It targets a higher level of line control in next-generation production processes as well as in existing lines.

The main flow of the FPD process is shown in Figure 1 [2]. Each piece of processing equipment in Figure 1 has a specialized role, such as cleaning, coating, proximity exposing, developing, etching, and resist stripping, and those pieces of equipment are connected in-line. Each piece of equipment usually has a number of buffers. Most production lines adopt a simple strategy to feed glass substrate into the first piece of equipment at a constant interarrival time, this being called the *tact time*. This strategy is simple and enables us to estimate the number of products that can be manufactured in a given time.

Due to solution foaming, chemicals, heat treating, and so forth, the processing time at each piece of equipment is uncertain and may vary according to the conditions at that time. If a sheet of substrate is sent to a piece of equipment that is still processing a previously sent sheet of substrate, that newly sent sheet can wait at a buffer in front of the piece of equipment. However, if, at that time, all buffers in front of the piece of equipment are occupied by previously sent sheets, there will be no buffer for the newly sent sheet to wait in. This phenomenon is called a *collision* between substrates. Collisions should be avoided whenever possible because glass substrate is expensive and fragile. Moreover, stopping the active system in the middle of production due to a collision could result in a significant increase in cost.

In the field of scheduling theory, a collision-like phenomenon is called a *blocking* in the flow shop model and is studied as an important factor to determine line efficiency. If the processing time is deterministic, many study results on blocking exist (see extensive survey in [3]). If the processing time is stochastic, study results are somewhat limited in comparison with their deterministic counterparts. For example, see [4, 5], where the purpose is to minimize the expected makespan. On the other hand, in queueing theory, depending on the rule for processing blockings (blocked calls cleared,

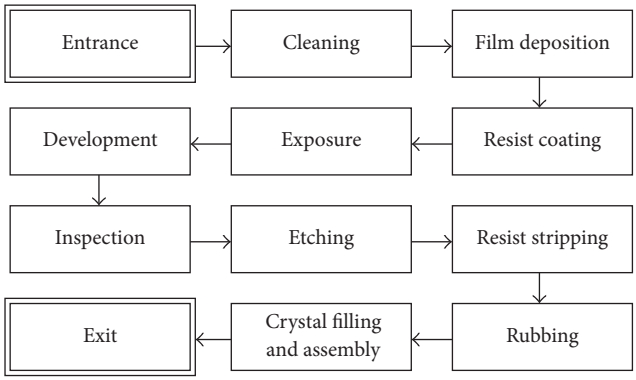


Figure 1: FPD process flow.

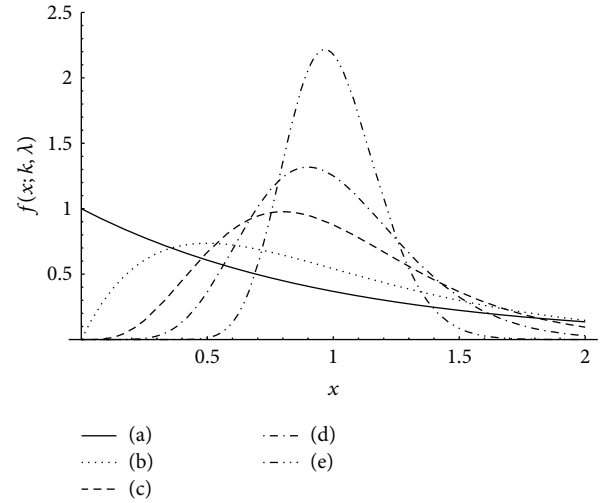


Figure 2: The pdf of the Erlang distribution: (a) $k = 1$, $\lambda = 1$ (exponential distribution); (b) $k = 2$, $\lambda = 2$; (c) $k = 5$, $\lambda = 5$; (d) $k = 10$, $\lambda = 10$; (e) $k = 30$, $\lambda = 30$.

blocked calls delayed, etc.), previous work mainly focused on performance measures in the steady state. In fact, we can find a lot of research on this in the field of queuing theory, for example, in [6–8].

In this paper, given the number of jobs to be processed in the prescribed tact time span, we focus on the measure recently presented in [9], which is the probability that there will be at least one collision, called the collision probability. In a comparatively new manufacturing system such as one manufacturing FPDs, the evaluation item (i.e., collision probability) is the new focus of observation.

The probability density function (pdf) of actual processing time is often represented by a bell-like curve such as in Figure 2(e). The pdf of normal distribution is also bell-shaped, but it has a weakness in that the pdf takes positive values in the negative domain, which is not true in the pdf of actual processing time. Therefore, in this paper, we assume that the processing time follows an Erlang distribution.

The pdf of Erlang distribution is defined as follows:

$$f(x; k, \lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} \quad \text{for } x \geq 0, \tag{1}$$

where two parameters λ and k are a positive real number and a positive integer, respectively. Its expectation and variance are given by k/λ and k/λ^2, respectively. Therefore, under Erlang distribution, we can set the expectation and the variance independently of each other by setting parameters λ and k appropriately. In Figure 2, five different probability density functions are plotted, where expectations of all cases are the same (i.e., one), but their variances decrease through cases (a)–(e) in Figure 2. Some of these pdfs are bell-shaped, and the pdf of Erlang distribution takes a value of zero for $x < 0$. Thus, Erlang distribution is flexible enough to represent actual processing times.

For the in-line machines model without buffers, studies on collision probability have been conducted. The evaluation item (i.e., collision probability) in the in-line machines model was first presented in [9]. It was shown in [9] that the collision probability can be approximately expressed by a multiple integration, assuming that the processing time of each piece of equipment follows general distribution. If the processing time follows Erlang distribution, a closed form formula of the approximate collision probability is derived without using any multiple integration [10]. A computer simulation method for computing the collision probability was also presented in [9]. This method can compute the probability for the processing time of general distribution. Note that those results obtained in [9, 10] do not consider buffers; that is, the number of buffers is assumed to be zero. In contrast, the in-line machine model in this paper considers buffers.

In a real FPD production line, in order to avoid collision between substrates, a number of buffers are placed in front of each piece of equipment. However, in reality, very few buffers seem to be used, and a large amount of money and space is wasted due to those that are not used. In designing an actual line, the number of buffers is often decided in advance from past experience. To the best of the authors' knowledge, no theoretical literature exists on the appropriate number of buffers to be used. Therefore, our research focuses on finding an optimal value for the number of buffers.

In this paper, we generalize the in-line machines model in [9, 10] to accommodate buffers. We present a computer simulation method for computing the collision probability in the model, in which each piece of equipment has an arbitrary number of buffers. This method is efficient as it computes in linear time whether a collision occurs or not at each piece of equipment. Next, we parameterize the collision probability and try to find a buffer allocation that achieves the smallest total number of buffers under an arbitrarily specified collision probability. The optimization problem is henceforth referred to as *the buffer allocation problem with collision probability*. Next, we present heuristics for this problem, characterized in that it computes a locally optimal solution. Finally, we implement the heuristics and confirm its performance from computational experimentation.

The remainder of this paper is organized as follows. In Section 2, we describe a formal model for the production line of FPDs. Then we formulate our problem as an optimization

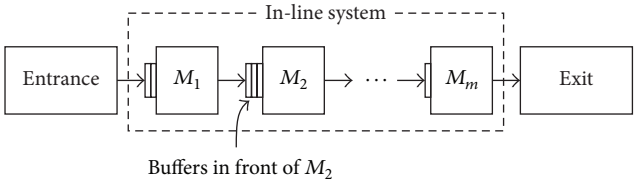


FIGURE 3: In-line machines model with buffers, where $b_1 = 2, b_2 = 3$, and $b_m = 1$.

problem. In Section 3, we consider how to derive start and finish times of each job at each piece of equipment, with each piece of equipment having an infinite number of buffers. In Section 4, we present a computer simulation method to evaluate the exact collision probability. In Section 5, based on the simulation method, we present heuristics to minimize the total number of buffers. In Section 6, we give some computational results and confirm that the heuristics computes good solutions in realistic computation time. Finally, Section 7 concludes this paper.

2. Model and Problem Formulation

We describe a formal model for the production line of FPDs. The following notations are used:

(i) $M_1, M_2, \ldots, M_m$: m different pieces of equipment in series. Each piece of equipment performs a dedicated role.

(ii) $J_1, J_2, \ldots, J_n$: n jobs to be processed.

(iii) $T_i^{(j)}$ (> 0): processing time of job J_i at M_j.

(iv) t_{tact} (> 0): tact time, that is, the time difference between the start time instants of J_i and J_{i+1} for all $1 \le i \le n-1$ at the entrance to the line.

(v) b_j $(\in \mathbb{Z}_+)$: the number of buffers in front of M_j, where $\mathbb{Z}_+$ denotes the nonnegative integer set.

The in-line machines model is illustrated in Figure 3. With the same time interval, t_{tact}, jobs are fed one by one into the line at the entrance. Each job is first processed on piece of equipment M_1. It is then automatically transported to the next piece of equipment M_2 after it has been finished on M_1. It is assumed, for simplicity, that the transportation time between pieces of equipment is nil. As soon as M_2 receives the job, it starts processing. In this manner, each job is processed on consecutive pieces of equipment in the order $M_1, M_2, \ldots, M_m$ and then sent to the exit. Moreover, we assume that the processing time $T_i^{(j)}$ at M_j is a random variable that follows Erlang distribution with parameters λ_j and k_j, and all $T_i^{(j)}$ $(1 \le i \le n, 1 \le j \le m)$ are independent of each other.

When a job J_i arrives at a piece of M_j, the system dynamics is as follows. If M_j is idle, it starts processing for J_i. Otherwise, if an empty buffer exists in front of M_j, then J_i waits at the buffer. When waiting, the queue discipline follows First-In-First-Out (FIFO), where all jobs are processed in the same order as they arrive in the queue. Since the number of buffers in front of M_j is b_j, the length of the queue can be, at most, b_j. In Figure 3, $b_1 = 2$, $b_2 = 3$, and $b_m = 1$. When all buffers in front of M_j are occupied (i.e., the length of the queue in front of M_j is b_j), if J_i arrives at M_j, then a collision occurs. The *collision probability* is the probability that there will be at least one collision.

The sequence of the number of buffers in front of each piece of equipment, that is, $(b_1, b_2, \ldots, b_m)$, is called a *buffer allocation*. It is called *feasible* if the collision probability is less than or equal to a given value α. Then our problem, that is, *buffer allocation problem with collision probability* (BAP-CP), is stated as follows. Given the number of jobs n, the number of pieces of equipment m, the tact time t_{tact}, the parameters of the Erlang distribution for each piece of equipment, and α $(0 \le \alpha \le 1)$ (specifying the collision probability), minimize the objective function $B = \sum_{j=1}^{m} b_j$ (i.e., total number of buffers) over the set of all feasible buffer allocations. An *optimal buffer allocation* is a feasible buffer allocation that achieves the smallest objective value.

3. Scheduling When the Number of Buffers Is Infinite

We consider a schedule under the assumption that the number of buffers in front of each piece of equipment is infinite. The schedule is a mapping $J_i \mapsto (st_i^{(j)}, ft_i^{(j)}: 1 \le j \le m)$, where the variables $st_i^{(j)}$ and $ft_i^{(j)}$ are defined as follows:

(i) $st_i^{(j)}$: time instant when job J_i is started on M_j,

(ii) $ft_i^{(j)}$: time instant when job J_i is finished on M_j.

Suppose that processing time $T_i^{(j)}$ is equal to $t_i^{(j)}$ for $1 \le i \le n$, $1 \le j \le m$. Then, using processing times $t_i^{(j)}$, we can compute a schedule as follows. J_i is started on M_j after the completion of two events, these being (i) J_i is finished on M_{j-1} and (ii) J_{i-1} is finished on M_j. After J_i is started on M_j, it is finished in $t_i^{(j)}$ unit times. Moreover, we can consider that each J_i is finished on M_0 at the time instant $(i-1)t_{\text{tact}}$ since each job is fed into the line at the entrance with t_{tact} interval. Therefore, the following recursive expressions hold:

$$st_i^{(j)} = \max\left\{ft_i^{(j-1)}, ft_{i-1}^{(j)}\right\} \quad (1 \le i \le n,\ 1 \le j \le m),$$

$$ft_i^{(j)} = \begin{cases} 0 & (i = 0,\ 1 \le j \le m), \\ (i-1)\,t_{\text{tact}} & (1 \le i \le n,\ j = 0), \\ st_i^{(j)} + t_i^{(j)} & (1 \le i \le n,\ 1 \le j \le m). \end{cases} \tag{2}$$

From (2), the time complexity for computing the schedule is $O(mn)$, as filling each entry requires $O(1)$ time. The obtained schedule is used to compute the collision probability in the next section.

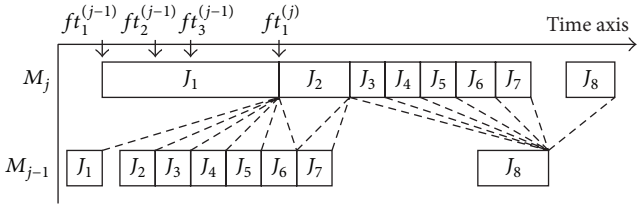


Figure 4: An example of the schedule at M_{j-1} and M_j when $n = 8$. The broken lines correspond to evaluations of the condition $ft_k^{(j-1)} < ft_i^{(j)}$.

4. Computing the Collision Probability

The number of buffers should be minimized in order to reduce the cost and space needed for manufacturing. However, the larger the decrease in the number of buffers is, the more the collision probability increases. A trade-off between them exists. When considering this trade-off, it is important to evaluate the collision probability under a given buffer allocation. Using the schedule obtained in the previous section, we present a simulation method to evaluate the exact collision probability.

Definition 1. A waiting sequence in front of M_j is a subsequence of jobs $(J_p, J_{p+1}, \ldots, J_q)$ for some $2 \le p \le q \le n$ such that

$$ft_k^{(j-1)} < ft_{p-1}^{(j)}, \tag{3}$$

for all $k = p, p+1, \ldots, q$. A waiting sequence $(J_p, J_{p+1}, \ldots, J_q)$ in front of M_j is called maximal if adding any job (J_{p-1} or J_{q+1}) would destroy that property. A maximal waiting sequence $(J_p, J_{p+1}, \ldots, J_q)$ in front of M_j, such that its length (i.e., $q-p+1$) is the largest over all maximal waiting sequences in front of M_j, is called maximum.

The $ft_k^{(j-1)} < ft_{p-1}^{(j)}$ in Definition 1 means that M_j is still processing J_{p-1} when each J_k ($k = p, \ldots, q$) arrives at M_j from M_{j-1}. Therefore, all jobs in the waiting sequence will wait at the buffers in front of M_j. However, a collision occurs if the length of the waiting sequence is larger than the number of buffers b_j. In other words, when the length of the maximum waiting sequence in front of M_j is lower than or equal to b_j, no collision will occur in front of M_j.

Using the example shown in Figure 4, we consider an efficient method to compute whether a collision at M_j occurs or not. In Figure 4, each $ft_i^{(j-1)}$ and $ft_i^{(j)}$ ($i = 1, 2, \ldots, n$) is given. When M_j is processing J_i, a waiting sequence in front of M_j can be computed as follows. Starting from $k = i$, k is incremented by one for each successive k. If the condition $ft_k^{(j-1)} < ft_i^{(j)}$ is true, the waiting sequence in front of M_j is $(J_{i+1}, \ldots, J_k)$. Therefore, if the length of the sequence (i.e., $k - i$) is larger than b_j, then a collision occurs; otherwise, k is incremented by one. On the other hand, if the condition $ft_k^{(j-1)} < ft_i^{(j)}$ is false, J_i is finished on M_j before J_k arrives at M_j. Therefore, when M_j is processing the next job J_{i+1} (i is incremented by one), we compute a waiting sequence in front of M_j. Note that, throughout this computation, the values of k and i are never decreased, and either k or i is incremented by one after the evaluation of the condition. Therefore, we can compute whether a collision at M_j occurs or not in at most $2n$ evaluations of the condition. In Figure 4, the waiting sequence $(J_3, J_4, J_5, J_6, J_7)$ is maximum. Therefore, the value of b_j needs to be at least five to avoid a collision.

From the above discussion, Pseudocode 1 computes whether a collision at M_j occurs or not. The computation time is $O(n)$.

```
Function COLLISION_CHECK(M_j)
Input: ft_i^(j-1), ft_i^(j) (1 ≤ i ≤ n) obtained from (2)
the number of jobs n, and the number of buffers b_j.
Question: Does at least one collision occur at M_j?
(1)  i := 1; k := 1;
(2)  while (i ≤ n && k ≤ n) {
(3)    if (ft_k^(j-1) < ft_i^(j)) {
(4)      if (b_j < k − i) return j;  // Collision occurs at M_j.
(5)      else k := k + 1;
(6)    } else i := i + 1;
(7)  }
(8)  return 0;  // Collision does not occur at M_j.
```

Pseudocode 1

Using COLLISION_CHECK(M_j), we can compute the collision probability in the in-line machines model as follows.

Algorithm 2 (SIMULATE).

Input. The input is the number of jobs n, the number of pieces of equipment m, the number of buffers b_j for all $1 \le j \le m$, the tact time t_{tact}, the parameters of the Erlang distribution for each piece of equipment, and a positive integer c (specifying the number of iterations, which is related to the accuracy for the collision probability).

Output. The output is the collision probability.

Step 1. loop := 1.

Step 2. Generate processing times $t_i^{(j)}$ ($1 \le i \le n$, $1 \le j \le m$) randomly from the Erlang distribution.

Step 3. Compute the schedule of each job using (2).

Step 4. Evaluate whether a collision occurs at any M_j by calling COLLISION_CHECK(M_j) for each $j = 1, \ldots, m$. Let loop := loop + 1. If loop $\le c$, return to Step 2; otherwise go to Step 5.

Step 5. Output the collision probability (the number of collisions evaluated in Step 4)/c.

For Steps 2, 3, and 4 in SIMULATE, the computation time is $O(mn)$. Since Steps 2, 3, and 4 are repeated c times, the computation time of SIMULATE is $O(cmn)$.

5. Heuristics for BAP-CP

In this section, we present heuristics for BAP-CP. The heuristics is composed of three stages. Stage 1 computes a buffer allocation $(\overline{b_1}, \overline{b_2}, \ldots, \overline{b_m})$ such that the collision probability is almost exactly zero from simulation. Therefore, the buffer allocation corresponds to an approximate upper bound on the optimal buffer allocation $(b_1^*, b_2^*, \ldots, b_m^*)$. In other words, $b_j^* \leq \overline{b_j}$ is assumed for each j.

Stage 1. Compute the buffer allocation $(\overline{b_1}, \overline{b_2}, \ldots, \overline{b_m})$.

Step 1. loop := 1, $\overline{b_j} := 0$ for each $j = 1, \ldots, m$.

Step 2. Generate processing times $t_i^{(j)}$ $(1 \leq i \leq n,\ 1 \leq j \leq m)$ randomly from the Erlang distribution.

Step 3. Compute the schedule of each job using (2).

Step 4. Compute the length of the maximum waiting sequence in front of each piece of equipment. For each M_j, if the length of the maximum waiting sequence is greater than $\overline{b_j}$, update the value of $\overline{b_j}$ to this length. Let loop := loop + 1. If loop $\leq c'$, return to Step 2.

Stage 1 is similar to SIMULATE. At Step 4 in Stage 1, the length of the maximum waiting sequence in front of M_j can be computed in a way similar to COLLISION_CHECK(M_j). Specifically, Steps 4 and 5 in COLLISION_CHECK(M_j) are rewritten as follows. When the condition $ft_k^{(j-1)} < ft_i^{(j)}$ is true, the length of the waiting sequence is $k - i$. Therefore, the maximum value of $k - i$ throughout the while loop corresponds to the length of the maximum waiting sequence in front of M_j. Thus, the length can be computed in linear time. At Step 4 in Stage 1, the c' is a positive integer constant for the computation of the buffer allocation $(\overline{b_1}, \overline{b_2}, \ldots, \overline{b_m})$. The value of c' is related to the accuracy for the buffer allocation $(\overline{b_1}, \overline{b_2}, \ldots, \overline{b_m})$.

Stage 2 computes a feasible buffer allocation $(b_1, b_2, \ldots, b_m)$ for BAP-CP. From Stage 1, the upper bound $\overline{b_j}$ on the number of buffers b_j for each j is obtained. Therefore, we search for the value of b_j for each j between 0 and $\overline{b_j}$, inclusive. For this, two variables left_j and right_j for each j are used, these being lower and upper bounds on b_j, respectively.

Stage 2. Compute the feasible buffer allocation $(b_1, b_2, \ldots, b_m)$.

Step 1. $\text{left}_j := 0$, $\text{right}_j := \overline{b_j}$ for each $j = 1, \ldots, m$.

Step 2. $b_j := 0$ for each $j = 1, \ldots, m$.

Step 3. With the current buffer allocation $(b_1, b_2, \ldots, b_m)$, compute the collision probability α' by using SIMULATE. Throughout SIMULATE, record the index k of M_k at which the most collisions occur.

Step 4. If $\alpha < \alpha'$ holds for the given collision probability α, then $\text{left}_k := b_k$, $b_k := \lceil (\text{left}_k + \text{right}_k)/2 \rceil$, and return to Step 3.

In Stage 2, starting from $b_j = 0$ for each j, the value of each b_j is increased until the current buffer allocation is feasible. When increasing the value of each b_j, we take a greedy approach. This means that the value of b_k is increased for M_k, the machine at which the most collisions occur. For the computation, the return value of COLLISION_CHECK(M_j) corresponds to the index of the piece of equipment at which a collision occurs. Note that the value of each b_j does not decrease throughout Stage 2. This may result in an excessive number of buffers for each M_j. To remedy this, we apply Stage 3.

Stage 3 computes an appropriate value of each b_k using the binary search method.

Stage 3. Improve the buffer allocation $(b_1, b_2, \ldots, b_m)$ computed in Stage 2.

Step 1. $k := 1$.

Step 2. $\text{right}_k := b_k$.

Step 3. $b_k := \lfloor (\text{left}_k + \text{right}_k)/2 \rfloor$.

Step 4. With the current buffer allocation $(b_1, b_2, \ldots, b_m)$, compute the collision probability α' by using SIMULATE.

Step 5. If $\alpha < \alpha'$, then $\text{left}_k := b_k + 1$; otherwise $\text{right}_k := b_k - 1$.

Step 6. If $\text{left}_k \leq \text{right}_k$, return to Step 3; otherwise $b_k := \text{left}_k$.

Step 7. $k := k + 1$. If $k \leq m$, return to Step 2.

The binary search for the value of b_k corresponds to Steps 2–6. The binary search is based on the property that the collision probability decreases monotonically with the value of b_k. After performing Stage 3, the obtained buffer allocation $(b_1, b_2, \ldots, b_m)$ is clearly feasible, and subtracting any value from any b_k would destroy the feasibility. Therefore, the obtained buffer allocation is locally optimal.

We analyze the time complexity of the heuristics. In Stage 1, the computation time is $O(mn)$ for Steps 2, 3, and 4. Since Steps 2, 3, and 4 in Stage 1 are repeated c' times, the computation time of Stage 1 is $O(c'mn)$. Next, the computation related to Step 3 in Stage 2 obviously dominates the total computation time for Stage 2. The total number of iterations of Step 3 is $O(\log \overline{b_1} + \log \overline{b_2} + \cdots + \log \overline{b_m}) = O(m \max_{1 \leq j \leq m} \log \overline{b_j})$. Since Step 3 costs $O(cmn)$, the total computation time of Stage 2 is $O(cm^2 n \max_{1 \leq j \leq m} \log \overline{b_j})$. Finally, the analysis for Stage 3 is the same as that of Stage 2. Therefore, the computation time of the heuristics is $O(c'mn + cm^2 n \max_{1 \leq j \leq m} \log \overline{b_j})$.

6. Computational Experiment

The heuristics is implemented on a PC (CPU: Intel Core i7 4 GHz, RAM: 32 GByte, OS: Windows 8.1 Professional). Throughout all simulations, we use Mersenne Twister [11] as

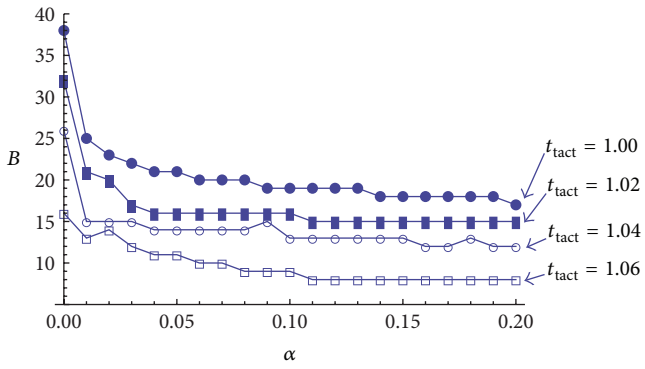


Figure 5: The relationship between α and B when $t_{tact} = 1.00, 1.02, 1.04$, and 1.06. Each marker corresponds to the result obtained by the heuristics.

the pseudorandom generator, and the number of iterations is set to $c = c' = 10,000$.

First, we survey the relationship between α (specifying the collision probability) and the total number of buffers B. For this, the number of jobs is set to $n = 100$, the number of pieces of equipment is set to $m = 8$, and the parameters of the Erlang distributions are set so that the expectation and variance of the processing time on each piece of equipment become equal to 1 and 0.01, respectively (i.e., $k = 100$, $\lambda = 100$ in (1)). The results obtained by the heuristics are shown in Figure 5. Figure 5 shows a tendency that the value of B decreases with an increasing value of α, clearly exhibiting the trade-off between α and B. Note that, although the value of B decreases sharply over small values of α, the value of B decreases by very small amounts when the value of α exceeds a certain threshold. Moreover, each buffer allocation shows a tendency that the values of b_j for pieces of equipment found earlier in the production line, such as M_1 and M_2, are a little larger than those for other pieces of equipment.

Figure 5 shows the results when $t_{tact} = 1.00$, 1.02, 1.04, and 1.06. We can confirm that the relationship between α and B is almost the same for each tact time, and the value of B decreases with an increasing value of t_{tact} when the value of α is fixed. The CPU time necessary for computing each marker in Figure 5 is at most 90 seconds.

Next, we survey the relationship between the tact time t_{tact} and the total number of buffers B when $\alpha = 0$. In a real FPD production line, collisions should be avoided whenever possible. For this, the parameter α corresponding to the collision probability is set to zero. The value of n is set to 100 and 1,000. The other parameters are the same as the settings in the first computational experiment (i.e., $m = 8$, $k = 100$, and $\lambda = 100$). The results obtained by the heuristics are shown in Figure 6. Figure 6(a) shows a tendency that the value of B decreases with an increasing value of t_{tact}.

We discuss a "good" value for tact time using Figure 6. Note that the value of B decreases by very small amounts when the value of t_{tact} exceeds a certain threshold t_{tact}^{upper}. In Figure 6(a), the value of t_{tact}^{upper} is about 1.2 and does not seem to depend on the value of n. The value of B should be minimized in order to reduce cost and space necessary for manufacturing. Moreover, the value of t_{tact} should be minimized in order to minimize the makespan. Therefore, any value of t_{tact} which is lower than or equal to the threshold t_{tact}^{upper} can be considered an effective value from the standpoint of the minimization of the value of t_{tact} under small values of B.

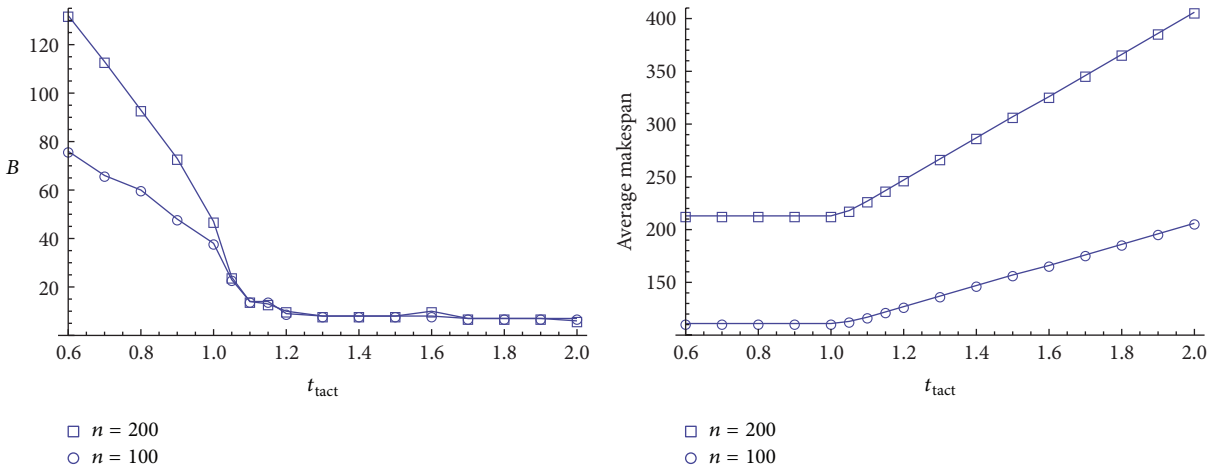
Table 1: The expectation of the processing time on each piece of equipment. For all types, the variance of the processing time on each piece of equipment is 0.01.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
Type 1	4	4	3	3	2	2	1	1
Type 2	1	2	3	4	4	3	2	1
Type 3	1	1	2	2	3	3	4	4

The average makespan corresponding to each marker in Figure 6(a) is shown in Figure 6(b). Note that the makespan is $ft_n^{(m)}$, which is computed from (2). The average makespan increases with an increasing value of t_{tact} when the value of t_{tact} is larger than a certain threshold t_{tact}^{lower}. It can be considered that the threshold t_{tact}^{lower} is equal to the largest value of the average processing times for all pieces of equipment. In Figure 6(b), the threshold t_{tact}^{lower} is 1.0 and does not seem to depend on the value of n. On the other hand, the average makespan remains almost constant when the value of t_{tact} is lower than t_{tact}^{lower}. The reason is that the length of the waiting sequence in front of M_k increases, where the expectation of the processing time at M_k is assumed to be the maximum. If there is more than one such M_k, then the index k is assumed to be the smallest index k among these M_k. As a result, the time instant $st_i^{(k)}$ at M_k for each J_i is almost the same when $t_{tact} < t_{tact}^{lower}$. (In the case of Figure 6(b), the value of k is one.) Therefore, any value of t_{tact} which is larger than or equal to the threshold t_{tact}^{lower} can be considered an effective value, since the makespan does not improve when $t_{tact} < t_{tact}^{lower}$. From the discussion related to Figure 6, any value of t_{tact} between t_{tact}^{lower} and t_{tact}^{upper} can be considered one of the most effective values from the viewpoint of cost, space, and makespan necessary for manufacturing.

Finally, we survey the relationship between t_{tact} and B in the cases of three types as in Table 1. The reason is that the expectation of the processing time is not always the same value for each piece of equipment in a real production line. The other parameters are as follows: $n = 100$, $m = 8$, and $\alpha = 0$. The results obtained by the heuristics are shown in Figure 7. From Figure 7, we could understand that there is not so much of a difference among the three types. However, Type 1's B value is the smallest when the value of t_{tact} is small. Therefore, when the value of t_{tact} is small, Type 1 is the most effective, then Type 2, and finally Type 3.

For each type, the average makespan is almost the same when t_{tact} is fixed. Throughout the three types, the average makespan is almost constant when $t_{tact} \leq 4$ and increases with an increasing value of t_{tact} when $t_{tact} \geq 4$. The reason is that, for each type, the largest value of the average processing times for all pieces of equipment is the same (i.e., four).



(a) (b)

FIGURE 6: (a) The relationship between t_{tact} and B when $\alpha = 0$. (b) The relationship between t_{tact} and the average makespan when $\alpha = 0$. Each marker corresponds to the result obtained by the heuristics.

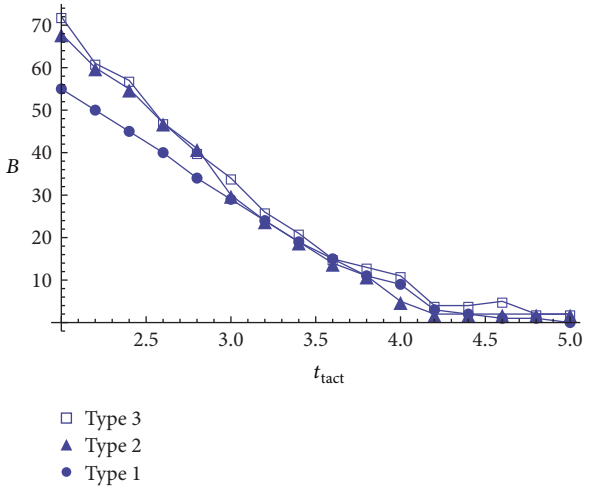


FIGURE 7: The relationship between t_{tact} and B when $\alpha = 0$. Each marker corresponds to the result obtained by the heuristics.

7. Conclusions

In this paper, we considered the in-line machines model with buffers. This is a generalized model of the known model in [9, 10]. We formulated the buffer allocation problem with collision probability. The problem has applications in the design of manufacturing systems. We also presented the heuristics to solve the problem. The heuristics computes a locally optimal buffer allocation in realistic computation time. Moreover, we implemented the heuristics and showed its performance through computational experimentation. We also discussed the effective values for tact time from a practical point of view. The discussion might be useful when designing production systems.

This paper presented the discussion in an in-line system. Discussion in a parallel or network system could be considered as one possible area of future work.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

References

[1] 2014, http://www.febacs.co.jp/eng/solution/crystal.html.

[2] 2014, http://www.screen.co.jp/eng/fpd/company/process.html.

[3] N. G. Hall and C. Sriskandarajah, "A survey of machine scheduling problems with blocking and no-wait in process," *Operations Research*, vol. 44, no. 3, pp. 510–525, 1996.

[4] M. Pinedo, "Minimizing the expected makespan in stochastic flow shops," *Operations Research*, vol. 30, no. 1, pp. 148–162, 1982.

[5] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Section 13.2, Prentice Hall, New York, NY, USA, 3rd edition, 2002.

[6] D. P. Heyman and M. J. Sobel, Eds., *Stochastic Models: Handbooks in Operations Research and Management Science*, vol. 2, North-Holland, Amsterdam, The Netherlands, 1990.

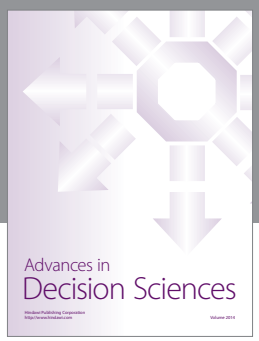
[7] L. Kleinrock, *Queueing Systems, Volume I: Theory*, Wiley Interscience, 1975.

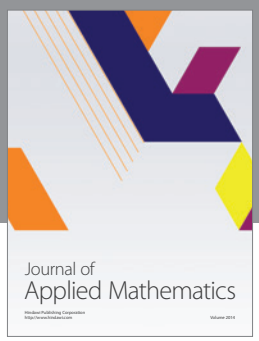
[8] K. Kunisawa and T. Honma, Eds., *Applied Queueing Dictionary*, Hirokawa Shoten, 1971.

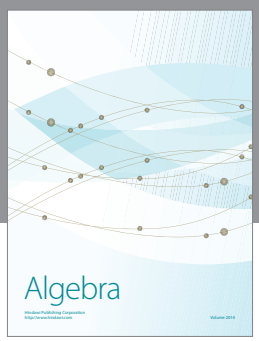
[9] E. Chiba, T. Asano, T. Miura, N. Katoh, and I. Mitsuka, "Collision probability in an in-line machines model," in *Transactions on Computational Science XIII*, vol. 6750 of *Lecture Notes in Computer Science*, pp. 1–12, Springer, Berlin, Germany, 2011.

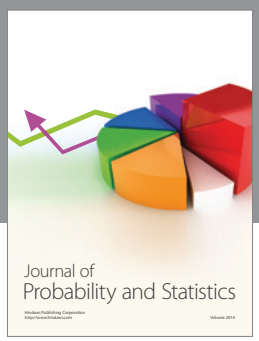
[10] E. Chiba, H. Fujiwara, Y. Sekiguchi, and T. Ibaraki, "Collision probability in an in-line equipment model under erlang distribution," *IEICE Transactions on Information and Systems*, vol. E96-D, no. 3, pp. 400–407, 2013.

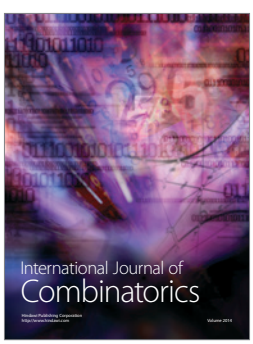
[11] 2014, http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html.

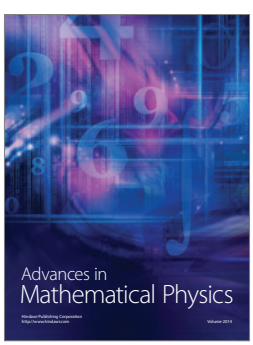


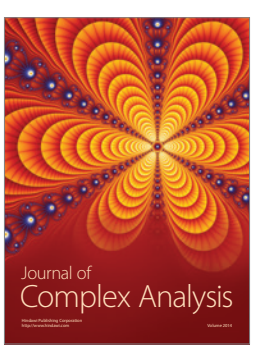


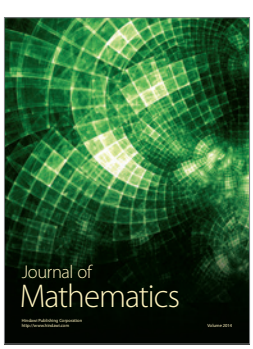


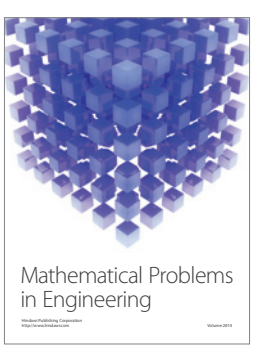


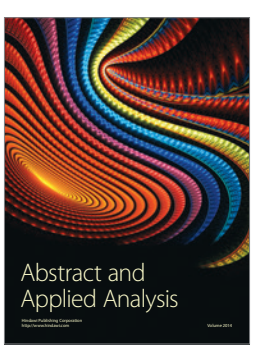


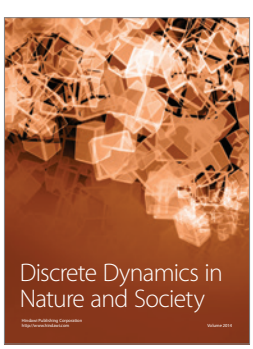


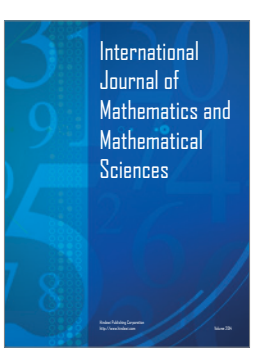












Hindawi

Submit your manuscripts at
http://www.hindawi.com