▩ 연구논문

유연 공정 라우팅에서의 고착 탐지 및 해결
Deadlock Detection and Resolution for Flexible Job Routing

임 동 순*
Yim, Dong-Soon
우 훈 식**
Woo, Hoon-Shik

Abstract

In order to resolve a deadlock problem in manufacturing systems, three main methods have been proposed—prevention, avoidance, and recovery. The prevention and avoidance methods require predicting deadlocks in advance in order to prohibit them. In contrast, the recovery method allows a system to enter a deadlock state, then resolves it usually using a common buffer. In this paper, a deadlock recovery method considering the impact of flexible job routings is proposed. This method is based on capacity-designated directed graph (CDG) model representing current requesting and occupying relations between jobs and resources in order to detect a deadlock and then recovers it.

1. Introduction

In operating and designing automated manufacturing systems, a deadlock has been frequently encountered as an important problem. A deadlock in a manufacturing system is a situation where job flow stops completely and cannot be restored simply by waiting [7,8]. The deadlock is also called a circular wait state, where a job is waiting for a resource being held by another job while occupying a resource that is, in turn, needed by another job [1]. When this waiting and occupying condition causes a chain reaction, the system falls into the circular wait state.

In order to handle a deadlock problem in manufacturing systems, three main methods have been proposed—prevention, avoidance, and recovery. While the deadlock prevention and avoidance methods prohibit deadlock, the recovery allows a system to enter a deadlock state, then resolves it usually using a common buffer. The avoidance method requires the detection of an *impending deadlock*. Under the impending deadlock state, the next job movements are possible but will eventually cause a deadlock. To detect the impending deadlock, future system states have to be precisely predicted. When a system allows jobs of flexible routings, however, the future state will not be easily predicted. There can be several alternatives in future states. This implies that obtaining a sufficient condition for impending deadlock is difficult. Therefore, deadlock avoidance strategies based on the

* 한남대학교 산업공학과
** 대전대학교 정보시스템공학과

necessary condition for an impending deadlock could be developed [13]. These strategies are somewhat conservative because they regard a non-deadlock state as a potential deadlock. Because of this conservative characteristic, the avoidance strategies tend to yield low utilization of equipment by extremely restricting the number of jobs in a system.

In a flexible job routing environment, the deadlock recovery will be more attractive because of its simplicity. The recovery requires the detection of current deadlock instead of impending deadlock. To determine whether the current system is in a state of deadlock or not, a model representing the waiting and occupying relations of jobs is required. Directed graphs [2,5,11], as well as Petri nets [1,3,10], have usually been used to model a system for the purpose of detection and avoidance. When all jobs in a system have fixed routings, a deadlock will simply be detected from the graph model. In the case of flexible routings, the number of next resources in which a waiting job will be processed may be more than one. By portraying these alternative resources for waiting jobs in a graph model, two types of deadlocks will be detected. These are classified as permanent deadlock and transient deadlock. Permanent deadlock refers a state where a set of resources and jobs are irrevocably blocked, whereas transient deadlock indicates that there is a potential for the deadlock to resolve itself in time [9]. When the permanent deadlock arises, a recovery action is always required. In the case of a transient deadlock, on the other hand, the decision on the recovery action has to be made. Once the action for recovery is decided, the deadlock can be resolved using a common buffer [4,6,7,12].

In this paper, a deadlock recovery method based on capacity-designated directed graph (CDG) [13] are developed as means to resolve a deadlock problem occurring in a flexible job routing environment. This method utilizes a dispatching or a scheduling decision for job movements. Under the recovery method, a real-time CDG model is created from the information on current states of the system. Once a permanent or a transient deadlock is detected in this model, it is resolved using one reserved space in common buffer as proposed in previous researches on deadlock recovery [9,12].

2. CDG^r models for deadlock detection

Capacity-designated directed graph (CDG) was designed to model a system for the purpose of deadlock detection, prevention, and avoidance. It is formally defined as a 4-tuple $G(N,A,M,C)$, where N represents a set of nodes, A represents arcs that connecting between nodes, M is the marking of N, and C represents the capacity of N. The number of current jobs in node n is represented as $M(n)$. In addition, the maximum number of jobs in node n is restricted to the capacity of $C(n)$.

Resources in automated manufacturing systems can be classified into processing, storing and transportation resources. In CDG models, processing and storing resources are represented as nodes. Originally, an arc connecting two nodes depicts the meaning of occupying and requesting. When a job occupying in the source node requests the destination node as a next processing resource, an arc connects both nodes.

A CDG^r model is created for the deadlock detection from the current system status. When a resource has a job requesting to move to the next resource, an arc is created to connect both resources. To complete the model for deadlock detection, the number of output arcs of a node should be at most one. Note that in a node whose capacity is more than one, several jobs may request to move to the next resource at the same time. If we acknowledge that more than one job in a resource cannot be transported at the same time, one job among requesting jobs at a resource should be selected for making the decision on the movement. Also, if fixed job routings are allowed in the system, the number of possible next resources for the selected job will be one. In this CDG^r, therefore, the number of output arcs from a node is assumed to be at most one.

The movement of the job in node n_i to node n_j, represented as an arc (n_i, n_j), is allowed under the condition that the capacity $C(n_j)$ is greater than the current number of jobs in n_j, i.e., $M(n_j)$. If $C(n_j)$ is equal to $M(n_j)$, the job cannot move to the node n_j. It is easy to derive the following necessary and sufficient condition for deadlock in CDG^r.

<u>Necessary and sufficient condition for deadlock in CDG^r</u>
A deadlock occurs in a CDG^r G=(N,A,M,C) if and only if there is a cycle, s, in G such that $\sum_{n_i \in S} M(n_i) = \sum_{n_i \in S} C(n_i)$

When flexible and alternative job routings are allowed, there can be more than one next resource for a waiting job. An extended CDG^r is considered in this paper so that these flexible routings are included in the model. In the extended CDG^r, a set of arcs is created connecting the resource in which a job is selected for movement to all possible next resources for the selected job. Therefore, there can be more than one output arc from a node. In this extended CDG^r, the above condition is no longer the necessary and sufficient condition for deadlock. The existence of the cycle satisfying the above condition indicates either a permanent deadlock or a transient deadlock. Permanent deadlock is what we call a general deadlock, whereas a transient deadlock, introduced at [9], is just a potential for deadlock. The following example illustrates the transient deadlock.

Example 1: Four machines (MC1, MC2, MC3 and MC4) process four types of jobs. Each job type has its own routing as shown in Table 1. For instance, job type 1 is processed at MC1, then processed at either MC2 or MC3. Finally, it is processed at MC1 again. Note that job types 1 and 3 have alternative routings. At each machine, an input buffer (IB) and an output buffer (OB) of capacity one are provided to temporarily store jobs. For a CDG^r, a machine and these input and output buffers are represented as one node. Since each machine processes one job at a time, the node has capacity of three. Figure 1 shows the current status of the system. A job currently occupying a resource is represented as p(*i,j*) where *i* is a job type and *j* is a current operation sequence. For instance, p(4,1) in output buffer of MC2 belongs to job type 4, and the operation sequence is 1. That is, p(4,1) is currently in MC2 for the first operation. An extended CDG^r is created in Figure 2. It is assumed that jobs p(1,1), p(4,1), p(3,1) in output buffers are currently requesting to move to the next resource. In this extended CDG^r, there is a cycle {MC1,MC2} in which the number of jobs equals the capacity. This cycle indicates a transient deadlock. If the next resource for job p(1,1) in the output buffer of MC1 is decided to be MC2, a deadlock will occur. Otherwise, the deadlock will be avoided by transporting the job to MC3 instead of MC2.

As shown in the above example, the transient deadlock state will evolve to either a deadlock or a non-deadlock according to the job assignment decision for the resource. It means that the transient deadlock can be resolved simply by an appropriate job assignment. In the case of permanent deadlock, however, a recovery procedure using a common buffer is required. The sufficient and necessary condition for permanent deadlock in an extended CDG^r model is obtained as follows.

Necessary and sufficient condition for permanent deadlock in extended CDG^r

A permanent deadlock occurs in an extended CDG^r G=(N,A,M,C) if and only if there is a set of cycles S in G satisfying the following condition:

Let $S=\{s_1, s_2, \ldots, s_m\}$, be a set of cycles in G satisfying the condition $\sum_{n_i \in S} M(n_i) = \sum_{n_i \in S} C(n_i)$. Then, every output nodes of any node in cycle, $s_k \in S$, is also included in cycle, $s_l \in S$.

Table 1. Job routings

Job Type	Routing
1	MC1,MC2/MC3,MC1
2	MC3,MC1
3	MC4,MC2/MC3
4	MC2,MC1

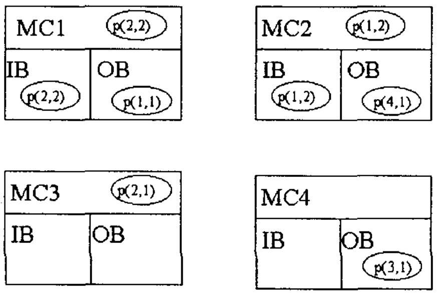

Figure 1. Current jobs in a manufacturing system

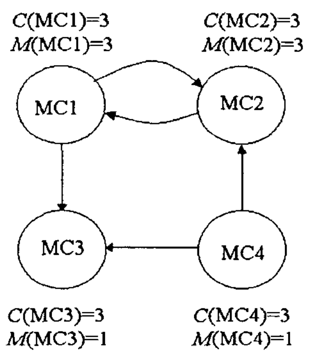

Figure 2. An extended CDG^r model

Proof: When a node has neither an input node nor an output node, it can be excluded for the consideration of a possible deadlock. Without loss of generality, therefore, we can assume that all nodes in CDG^r have both input and output nodes. It means that all nodes in CDG^r are included in one or more cycles. Assume that a job in node n_i of an extended CDG^r is in a deadlock state. Let $N_D = \{ N_{D_1}, N_{D_2}, \ldots, N_{D_p}\}$ be a set of nodes of G in which the current number of jobs is equal to the capacity. Then, there is not a path from n_i to $n_l \in N_D$, where the output node of n_l is not in N_D. If there is such a path, the job in n_i is not in a deadlock state since the job in n_i can move to the output node along the path after the job in n_l moves to its output node. Therefore, all output nodes of n_i are in N_D, and any path starting from an output node of n_i is included in N_D. Since all nodes in N_D have both input and output nodes, the paths constitute cycles. In other word, there are cycles in which all nodes are in N_D, and any output node connected from the cycles is included in N_D.

The proof for the opposite direction is evident. If all output nodes from a node n_i in cycle s_k are included in nodes of S, the job in n_i cannot move to its output nodes. When this condition applies to all jobs in nodes of S, a permanent deadlock arises.

3. Deadlock recovery method using CDG^r model

In this section, a deadlock recovery method considering the impact of flexible job routings is proposed. The deadlock recovery method detects either a permanent deadlock or a transient deadlock from an extended CDG^r model, then resolves it using one reserved space in a common buffer. Specifically, the deadlock is resolved by moving the deadlocked job from the machine to a common buffer and reallocating the released machine to a waiting job. This procedure assumes that one buffer space is reserved exclusively for the recovery [9,12].

As stated before, it is assumed that a priority list for job movements considering the flexible job routings is determined by a decision module. Also, it is assumed that at a resource, at most one job can request a movement to the next resource. Let $\{(p_i, n_i), i = 1, 2, \ldots, m\}$ be a priority list for job movements, where p_i is a waiting job and n_i is the next machine for p_i. Also, $n(p_i)$ and $p(n_i)$ are the current machine of job p_i and the job requesting movement at machine n_i respectively. By representing each element in the priority list as an arc connecting from $n(p_i)$ to n_i, an extended CDG^r is created. In this extended CDG^r, the following several cases regarding job movements exist.

Case 1) There is no waiting job.

Case 2) There is a waiting job that can move to the next resource immediately.

Case 3) All waiting jobs cannot move to the next resource immediately.

3-1) There is cycles $\{ s_i, i = 1, 2, \ldots, m\}$, which satisfy condition

$\sum_{n_i \in S} M(n_i) = \sum_{n_i \in S} C(n_i)$.

3-1-1) There is cycles that satisfy the necessary and sufficient condition of permanent deadlock.

3-1-2) All cycles does not satisfy the necessary and sufficient condition of permanent deadlock.

3-2) There is no cycle that satisfies the condition $\sum_{n_i \in S} M(n_i) = \sum_{n_i \in S} C(n_i)$.

Case 1 is trivial since no job will be considered for movement. In case 2, the waiting job can be transported to the next resource. In case 3, however, the detection of a deadlock will be required to identify the deadlocked jobs. Notice that only case 3-1-1 corresponds to a permanent deadlock state. Figure 3 shows several extended CDG^r models to illustrate case 3. In this figure, all nodes are assumed to have a capacity of one. When a node has a job in a processing state, it is represented as a dark circle. As depicted in Figure 3-a), case 3-2 arises when there is a path such that $M(n_i)$ equals to $C(n_i)$ for each node n_i in the path. Job p_1 in node n_1 and job p_2 in node n_2 are currently requesting movement to their next resource. However, p_3 in node n_3 has not yet requested movement to the next resource. When p_3 is presently in a processing state, this will happen. In this case, waiting jobs p_1 and p_2 cannot move to the next resource. The only way is waiting until the system state changes. Figure 3-b) illustrates the case 3-1-2. This case arises when there is a path that starts from a node in a cycle s_i and ends at a node that is not included in any cycle $\{s_i, i=1,2,\ldots,m\}$. In Figure 3-b), a path $\{n_2, n_4\}$ belongs to such a path. The number of jobs in the terminal node n_4 of the path will equal the capacity of that node. Since there is no output arc from the node n_4, the job in that node has not yet requested the movement to the next resource. This state belongs to the transient deadlock. Even though this state is not a permanent deadlock, it will be desirable to move waiting jobs p_1, p_2 and p_3 in the cycle to their next resource using a common buffer of capacity one rather than waiting until a system state changes as in the case of 3-2. Figure 3-c) illustrates the case 3-1-1. Two cycles satisfy the necessary and sufficient condition of permanent deadlock. In resolving this deadlock state, a cycle should be selected. Notice that several cycles satisfying the condition $\sum_{n_i \in S} M(n_i) = \sum_{n_i \in S} C(n_i)$ may also exist in case 3-1-2. Some decision module will be required to perform the selection of a cycle for deadlock resolution. Once a cycle is selected, the movement of waiting jobs in the cycle can be performed using one reserved space at common buffer.

The following algorithm describes the procedure for the deadlock recovery method based on extended CDG^r models. The algorithm first finds a job falling under case 1 to transport the job to the next resource. Notice that the job movement is possible only when the next resource has a space at the input interface for the entering job. When such a job is not found, the algorithm detects a deadlock and resolves it using a common buffer. From all cycles that belong to case 3-1-1 (i.e., permanent deadlock) and

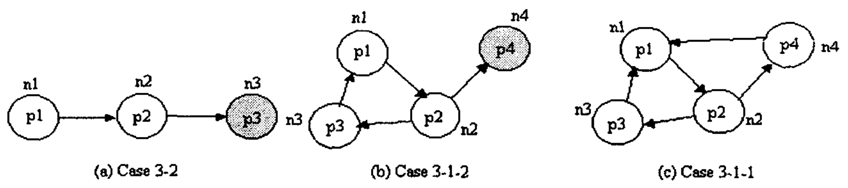

Figure 3. Extended CDG^r models that belong to case 3

case 3-1-2 (i.e., transient deadlock), a cycle is selected to perform movement of waiting jobs in the cycle.

<u>**Algorithm: RecoveryBasedOnCDG^r**</u>

Input: A priority list $D=\{(p_i, n_i), i=1,2,\ldots,m\}$ for job movements

```
// movement of a waiting job to the next resource which has a space at an input interface
1   for $i \leftarrow 1,2,\ldots,m$ do
2       if resource $n_i$ has a space at the input interface then
3           perform a movement of $p_i$ to $n_i$ and stop
4       endif
5   endfor
// creation of an arc list $L$
6   for $i \leftarrow 1,2,\ldots,m$ do
7       if resource $n_i$ has a waiting job then
8           if resource $n_i$ will have a space at the input interface by transporting
                the waiting job in the resource then
9               add $\{n(p_i), n_i\}$ to arc list $L$
10          endif
11      endif
12  endfor
// Movement of waiting jobs in deadlock state
13  find all cycles from arc list $L$
14  if no cycle is found then stop
15  else select a cycle ( $n_{k1}, n_{k2}, \ldots, n_{km}$) to move waiting jobs in each node
16      perform a movement of waiting jobs in the following order
        1) perform a movement of $p(n_{km})$ to common buffer
        2) perform a movement of $p(n_{km-1})$ to $n_{km}$
        3) perform a movement of $p(n_{km-2})$ to $n_{km-1}$
        .
        km) perform a movement of $p(n_{k1})$ to $n_{k2}$
        km+1) perform a movement of a job moved to the common buffer at step 1) to $n_{k1}$
17  endif
```

The complexity of the algorithm depends on computation times to find cycles. To find a cycle in a graph, a depth-first or a breadth-first search can be exploited. When elements in the priority list represent arcs in a graph, the complexity of finding all cycles is $O(n \times (m/n)!)$, where m is the number of elements in the list, and n is the number of nodes.

4. Simulation experiments

A simulation of a flexible manufacturing cell (FMC) was conducted to analyze the performance of the proposed deadlock recovery method. The FMC consists of four machines, an input conveyor, an output conveyor, and a robot. When a job needs to be transported for the next process, it requests the movement to the robot. Then, the robot is responsible to transport waiting jobs. When more than one movement is simultaneously requested, a priority list for the movements is created. An early entering job into the cell has high priority. When the number of next resources for a job is more than one, the priority for the next resource is randomly given. This dispatching rule is arbitrarily chosen for the analysis of deadlock recovery method. Four job types as shown in Table 1 are processed. The processing time at each machine is assumed to be constant with 10 minutes. Also, it is assumed that there are always waiting jobs at the input conveyor. In addition, each job type is randomly generated with an equal probability.

In evaluating the CDG^r-based deadlock recovery method, the performance is compared with a static deadlock avoidance method proposed in literature [13]. In the CDG model for the static deadlock avoidance method as shown in Figure 4, the common buffer is added as a node. Also, the capacity of the common buffer is set from 1 to 5 to analyze the effect of buffer size. Notice that the job route from MC1 to MC3 is represented as two arcs passing through the common buffer in order to resolve the deadlock, i.e., (MC1, CB) and (CB, MC3). By applying this arc replacement mechanism, each arc in the original CDG without common buffer is replaced with two arcs in the new CDG in which a common buffer is added as a node to resolve the deadlock. In Figure 4, three minimal cycles exist. According to the cycle reduction mechanism of CDG described as in literature [13], each cycle is reduced into a macro node. Also, the capacity of each macro node is determined as in Table 2.

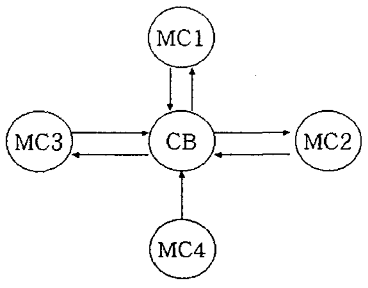

Figure 6. A new CDG model for static deadlock avoidance

Table 2. Macro nodes in new CDG for static deadlock avoidance

Macro node	Nodes included in a macro node	Capacity
s_1	MC3,CB	C(MC3)+Cb-1
s_2	MC1,CB	C(MC1)+Cb-1
s_3	MC2,CB	C(MC2)+Cb-1
s_4	MC1,MC3,CB	C(MC1)+C(MC3)+Cb-2
s_5	MC1,MC2,CB	C(MC1)+C(MC2)+Cb-2
s_6	MC1,MC2,MC3,CB	C(MC2)+C(MC3)+Cb-2
s_7	MC1,MC2,MC3,CB	C(MC1)+C(MC2)+C(MC3)+Cb-3

Table 3. Simulation results of deadlock resolution methods

Deadlock resolution methods		Recovery	Static avoidance method				
Capacity of common buffer		1	1	2	3	4	5
Avg. work-in-process (# of jobs)		3.62	2.37	3.14	3.82	4.65	5.40
Avg. flowtime (min)		106.74	118.96	102.61	115.58	134.00	148.26
Avg. throughput rate (# of jobs/480 min)		20.4	13.5	19.2	19.7	19.8	19.8
Avg. utilization (%)	MC1	0.40	0.27	0.37	0.38	0.38	0.38
	MC2	0.26	0.21	0.25	0.26	0.24	0.24
	MC3	0.17	0.08	0.16	0.15	0.18	0.18
	MC4	0.13	0.08	0.12	0.12	0.12	0.12

Table 3 shows the simulation results for both deadlock resolution methods, It is shown that the system performance of deadlock avoidance method does not improve in case that the capacity of common buffer is larger than three compared with deadlock recovery method. When the capacity is less than three, starvation frequently occurs at each machine. Even if the starvation occurrences decrease as the capacity increases over three, average flow time increases noticeably, and average throughput rate does not improve because of frequent blocking. Compared with the static deadlock avoidance method, the CDG^r-based recovery method produces better performance in general.

5. Concluding remarks

In operating automated manufacturing systems, a deadlock caused by inappropriate job assignments to resources can frequently occur. In deciding a job assignment, job routings play an important role. When jobs have flexible routings, there will be a number of alternative resources to which a waiting job can be assigned. At a glance, it seems to provide more chance for a deadlock-free system. This flexibility, however, makes the deadlock problem difficult to be tackled. The prediction of a future system state that is required for deadlock prevention and avoidance methods becomes complex. Although an avoidance method based on the necessary condition for an impending deadlock does not require precise prediction of a future system state, it tends to restrict the number of jobs in a system. Hence, the system performance will be undesirable.

In this paper, a deadlock recovery methods based on capacity-designated directed graph model is proposed. In order to detect deadlocks, an extended CDG^r model is created dynamically. This CDG^r model includes only the information on the immediate next routes of current jobs in the system. The detected deadlock is resolved by using one reserved space in a common buffer.

A simulation study shows that the recovery method based on CDG^r model is more efficient than the static deadlock avoidance method. The static deadlock avoidance method does not need to create a real-time CDG model. Instead, only one CDG model is created at the planning phase. However, the conservative nature of the deadlock avoidance method usually requires a common buffer with a large capacity in order to lessen the starvation occurrences at machines.

References

[1] Banaszak, Z. A. and Krogh, B. H., 1990, Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows, *IEEE Transactions on Robotics and Automation*, **6/6**, 724-734.

[2] Cho, Hyeunbo, Kumaran, T. K. and Wysk, Richard A., 1995, Graph-Theoretic Deadlock Detection and Resolution for Flexible Manufacturing Systems, *IEEE Transactions on Robotics and Automation*, 11/3, 413-421.

[3] D'Souza, K. A., 1994, A Control Model for Detecting Deadlocks in an Automated Machining Cell, *Computers & Industrial Engineering*, **26/1**, 133-139.

[4] Egbelu, P. J. and Tanchoco, J. N. A., 1984, Characterization of AGV Dispatching Rules, *International Journal of Production Research*, **22/3**, 359-374.

[5] Kumaran, T. K., Chang, W., Cho, H. and Wysk, R. A., 1994, A Structured Approach to Deadlock Detection, Avoidance and Resolution in Flexible Manufacturing Systems, *International Journal of Production Research*, **32/10**, 2361-2379.

[6] Leung, Y. T. and Shen, G., 1993, Resolving Deadlocks in Flexible Manufacturing Cells, *Journal of Manufacturing Systems*, **12/4**, 291-304.

[7] Okogbaa, O. G. and Huang, J., 1992, A Simulation Study of Deadlock Avoidance in FMS, *1st Industrial Engineering Research Conference*, 197-201.

[8] Tempelmeier, H and Kuhn, H., 1993, *Flexible Manufacturing Systems: Decision Support for Decision and Operation* (New York, NY: John Wiley & Sons).

[9] Venkatesh, S. and Smith, J. S., 1998, Deadlock Detection and Resolution in the Real-time Control of Flexible Manufacturing Cells, Working Paper, Texas A & M University, College Station, Texas.

[10] Viswanadham, N., Narahari, Y. and Johnson, T. L., 1990, Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models, *IEEE Transactions on Robotics and Automation*, **6/6**, 713-723.

[11] Wysk, R. A., Yang, N. S. and Joshi, S., 1991, Detection of Deadlocks in Flexible Manufacturing Cells, *IEEE Transactions on Robotics and Automation*, **7/6**, 853-859.

[12] Yang, N. S., 1989, *Resolution of System Deadlocks in the Real-time Control of Flexible Manufacturing Systems*, Unpublished Doctoral Dissertation, Pennsylvania State University, State College, Pennsylvania.

[13] Yim, D. S., Kim, J. I., and Woo, H. S., 1997, Avoidance of Deadlocks in Flexible Manufacturing Systems using a Capacity-Designated Directed Graph, *International Journal of Production Research*, **35/9**, 2459-2475.