Int J Adv Manuf Technol (2005) 26: 659–668
DOI 10.1007/s00170-003-2020-4

ORIGINAL ARTICLE

Jung-woon Yoo · Eok-Su Sim · Chengxuan Cao · Jin-Woo Park

An algorithm for deadlock avoidance in an AGV System

Received: 26 September 2003 / Accepted: 4 November 2003 / Published online: 1 December 2004

Abstract In this paper, a simple and easily adaptable deadlock avoidance algorithm for an automated guided vehicle (AGV) system is presented. This algorithm uses the graph-theoretic approach. Unlike Petri-net-based methods, which are complex and static, it is easy to modify the existing model as the configuration of the system changes. Therefore, it is suitable for the AGV system in a flexible manufacturing system (FMS) and a retail or postal distribution center. Moreover, because it is very simple, it is appropriate for real-time control mechanisms.

This paper consists of two parts: the first part presents an AGV deadlock avoidance algorithm that uses the graph-theoretic approach, and the second suggests appropriate routing strategies based on the proposed algorithm. The results show that this deadlock avoidance algorithm can be modified easily whenever the configuration of an FMS changes and provide high-performance on the deadlock avoidance. Finally, experimental results that confirm the validity of this approach are provided.

Keywords AGV · Conflict free · Deadlock avoidance · Graph theory · Path matrix

1 Introduction

There have been many reports on such automation systems as material handling and distribution systems and flexible manufacturing systems (FMSs). Among the various research areas in automation systems, one area that has often been overlooked in previous studies on the design and operation of automation systems is the phenomenon known as "deadlock". Most automation systems use the deadlock prevention algorithm (DPA) to prevent deadlocks, which is the most inefficient approach for preventing deadlocks. The deadlock prevention method prohibits the system from deadlock in the design stage (i.e., deadlock prevention in the scheduling algorithm). Therefore, there has been little need for solving real-time deadlock problems.

Currently, as the market is getting more and more competitive and endlessly demanding various services and products, it requests a high level performance from the automation systems. The methods using deadlock-free scheduling algorithms and/or additional facilities (i.e., detours) are inappropriate, because they cause lower utilization. Therefore, there is a need for a more efficient real-time deadlock-free control algorithm.

As the market tends towards infinite competition, not only automation but also flexibility is recognized as essential factors. However, most of the deadlock-free methods do not take system flexibility into consideration [2–4]. In the case of Petri-net approach, when a small change occurs in a physical layout, the model for a deadlock-free operation is of no use. The deadlock-free approach must recommence from the system modeling. Therefore, it is absolutely necessary to provide a deadlock-free approach, which holds regardless of the system changes.

The above situation is still the case in flexible manufacturing systems and material handling and distribution systems. When it comes to postal distribution centers, together with the growth of e-commerce, the amount of mail, which includes letters, packets, and parcels, is growing more and more and fluctuates day by day. In addition, it is almost impossible to forecast the destination of the mail. Consequently, for the sake of meeting lead time requirements, it is indispensable for a postal system to introduce flexible automation systems, one of which is the postal AGV system that this paper intends to deal with.

This paper is organized as follows: Sect. 2 outlines the deadlock detection method based on graph theory and compares it with other methods. Section 3 presents the deadlock avoidance algorithm (DAA) in the case of 2 AGVs and then verifies the

J.-W. Yoo
e-Logistics Research Team,
Electronics and Telecommunications Research Institute,
161 Gajeong-dong, Yuseong-gu, Daejeon 305-350, Korea

E.-S. Sim · J.-W. Park (✉)
Seoul National University,
San 56-1, Shillim-dong, Kwanak-gu, Seoul 151-742, Korea
E-mail: autofact@snu.ac.kr

C. Cao
School of Management,
University of Science and Technology Beijing,
Beijing 100083, P.R. China

DAA by solving some case examples. Section 4 shows the experiment design and its results. Finally, conclusions and suggestions on further research are presented in Sect. 5.

2 Deadlock detection method

2.1 Graph-theoretic cycle detection method

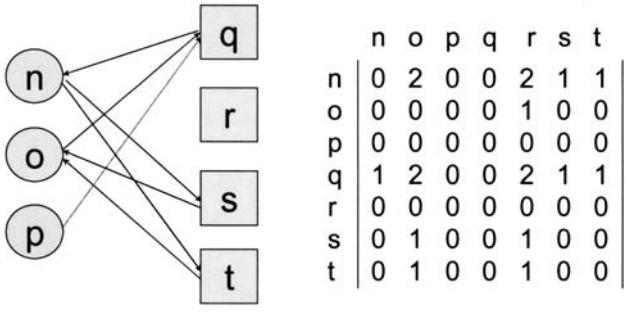
The Resource-allocation graph is recognized as a powerful tool that can describe deadlocks precisely**?**[1, 5]. This graph is a bipartite digraph of the equation, $G = (V, E)$, which consists of a set of vertices $V = \{1, 2, \ldots, |V|\}$ and a set of edges $E = V \times V$. The set of vertices V is composed of two kinds of resources, $A = \{a_1, a_2, \ldots, a_n\}$, which is the set consisting of all AGVs in the system, and $Z = \{z_1, z_2, \ldots, z_m\}$, which is the set consisting of all zones in the AGV path layout. A path from u to w, $u, w \in V$ is a sequence of nodes $v_0, v_1, v_2, \ldots, v_k$, such that $v_0 = u$, $v_k = w$ and $(v_i, v_{i+1}) \in E$ for $0 \leqslant i < k$. A cycle is defined as a path from u to w, where $u = w$ and the path has at least two edges. On the other hand, a graph is acyclic if it contains no cycles.

For a more detailed explanation, the edges comprise two types of edge: the requested edge and the assigned edge. The requested edge, $a_i \rightarrow z_j$, is a directed edge from the AGV a_i to zone z_j, and it implies that the AGV a_i has requested zone z_j and is currently waiting for that zone to reserve. On the contrary, the assigned edge $z_j \rightarrow a_i$, is a directed edge from zone z_j to AGV a_i, which means that the zone z_j has been already allocated or reserved to the AGV a_i. Since a AGV cannot exist on more than two zones simultaneously and a zone cannot receive more than two AGVs coincidently, there is only one assigned edge $z_j \rightarrow a_i$ for each zone z_j. Figure 1 shows a resource-allocation graph of its corresponding AGV system, where a cycle means a deadlock.

We define the path matrix representation of a graph G as a $|V| \times |V|$ matrix $P = [i, j]$ $1 \leqslant i, j \leqslant |V|$ with,

$$[i, j] = \begin{cases} l & \text{if there are } l \geqslant 1 \text{ different paths from } i \text{ to } j \\ 0 & \text{if there are } l < 1 \text{ different paths from } i \text{ to } j \end{cases}$$

We denote a column j of a path matrix P by $P[*, j]$ and a row i by $P[i, *]$ and the cross product between the column vector and the row vector is defined as $P[*, j] \otimes P[i, *]$. Furthermore, we denote an identity column vector of size $|V|$ by I_j, which consists of zero elements anywhere except at element i, and denote the transpose of I_j by I_j^T. Figure 1b shows the path matrix corresponding to its resource-allocation graph in Fig. 1a.



	n	o	p	q	r	s	t
n	0	2	0	0	2	1	1
o	0	0	0	0	1	0	0
p	0	0	0	0	0	0	0
q	1	2	0	0	2	1	1
r	0	0	0	0	0	0	0
s	0	1	0	0	1	0	0
t	0	1	0	0	1	0	0

(a) **Resource-allocation graph** (b) **Path matrix**

Fig. 1. A resource allocation graph and its corresponding path matrix

We need the dynamic change of the path matrix whenever an edge is inserted or deleted as shown in Fig. 1. Let's consider the situation where an edge (u, v) is inserted. Then, under the condition of the path matrix P of an acyclic digraph G, the number of paths passing through the edge (u, v) forms a path matrix $P_{(u,v)}$, which is defined as the outer product of the column vector $P[*, v] + I_v$ and the row vector $P[u, *] + I_u^T$, that is,

$$P_{(u,v)} = (P[*, v] + I_v) \otimes \left(P[u, *] + I_u^T \right).$$

The number of paths reaching node u corresponds to the column vector $P[*, u]$, and the number of paths starting from node v corresponds to the row vector $P[v, *]$. Since the edge (u, v) is newly inserted, we should take it into consideration that there are paths just starting at u or just ending at v. Semantically, the identity vector I_u means the paths starting at u and the I'^T_v corresponds to the paths ending at v. The outer product means all the newly created paths passing through an edge (u, v). That is, each element of $P_{(u,v)}[i, j]$ is the product of the number of paths reaching u from i and the number of paths ending j from v, where $1 \leqslant i, j \leqslant |V|$. Adding $P_{(u,v)}$ to the current path matrix P represents the changed path matrix by the insert of an edge (u, v). In the case of deleting an edge (u, v), instead of adding $P_{(u,v)}$ to the current path matrix P, we subtract $P_{(u,v)}$ from the current path matrix P. Throughout these matrix calculation procedures, the number of paths can be kept in the path matrix dynamically. The following example shows $P_{(u,v)}$ and new path matrix P^{new} in the case where edge (u, v) is inserted.

$$P_{(u,v)} = \left[\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right] \otimes \begin{bmatrix} (1, 2, 0, 0, 2, 1, 1) \\ +(0, 0, 0, 1, 0, 0, 0) \end{bmatrix}$$

$$P^{new} = P^{old} + P_{(u,v)} = \begin{bmatrix} 0 & 2 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 2 & 1 & 1 \\ 1 & 2 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Then, how can the deadlock, in other words, cycles in the resource allocation graph, be identified from the path matrix, when inserting an edge (u, v). It is sufficient to check whether the sum of diagonal elements of the path matrix is zero or not, which means there are no cycles.

2.2 Petri-net-based deadlock detection method

The Petri-net-based modeling technique has been frequently used to detect deadlock. It has many useful characteristics in representing the current state and in generating future states to analyse the deadlock in AGV systems. Its characteristics are given as follows [6]: graphical representation, concurrent and dynamic description, state generation, conflicting representation.

A Petri net is a five-tuple graph $\langle P, T, I, O, M_o \rangle$, where $P = \{p_1, p_2, \ldots, p_m\}$ denotes a finite set of places, where m refers to the total number of places in the PN, and $T = \{t_1, t_2, \ldots, t_n\}$ denotes a finite set of transitions, where n refers to the total number of transitions in the PN. $I : (P \times T) \rightarrow N$ is the input function where $N = \{0, 1, 2, 3, \ldots\}$. $O : (P \times N) \rightarrow N$ is the output function. M_o is the initial marking of the PN. A PN with tokens distributed in some places is also known as a marked PN.

Now the deadlock detection method that uses a Petri net can be introduced. First of all, the object system is described using a Petri-net formalism, and then the system states are generated by activating certain events (or transitions). In the middle of a transition, it may be possible to arrive at a certain odd state that has no enabled transition, i.e., infeasible marking. This state is the deadlock state. In this way, deadlock situations can be detected.

2.3 Comparison between the two deadlock detection methods

The advantage of the graph-theoretic approach is its status-modeling flexibility. This approach has great advantages in two aspects. One is an easy system modeling; the other is an easy model modification.

From the view-point of system modeling, this approach can represent the corresponding systems easily and quickly.

Figure 2 describes the object system. The comparison of Figs. 3 and 4 shows how easily this approach can model a system.

As shown in Fig. 4, the Petri-net model is quite complex. Hence, system modeling using a Petri net is a time-consuming task. Furthermore, if the systems layouts are altered, the Petri-net model must be redesigned. This requires excessive time and effort.

From the viewpoint of modeling modification, this approach can modify the existing model freely. Figure 5 represents a situation, where an AGV is added to the system and the layout is slightly changed (zone 4 is created). Even though the system configuration is altered, there is little change in the model.

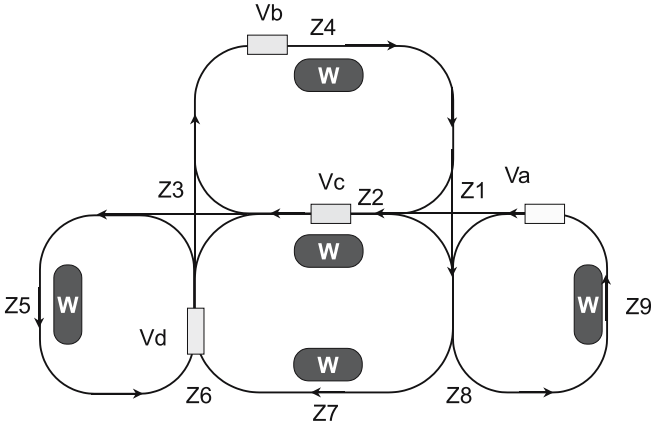


Fig. 2. An object system example for modeling

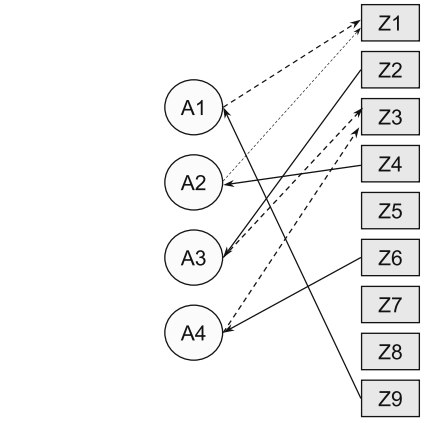


Fig. 3. A model of the graph-theoretic approach

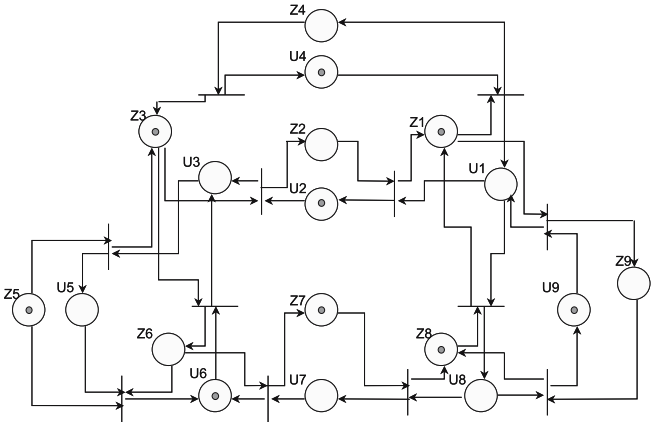


Fig. 4. A model of Petri net approach

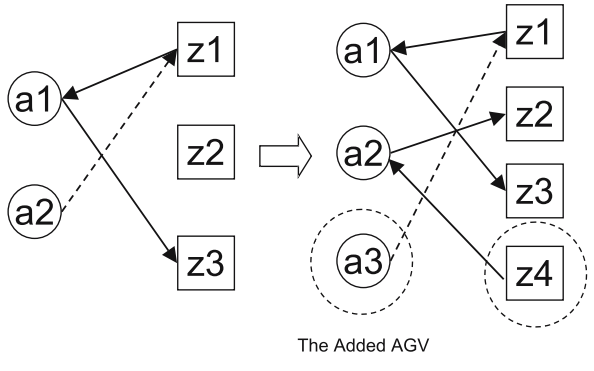


Fig. 5. The example of a model modification

3 Deadlock avoidance method

Figure 6 shows a certain pilot FMS plant, which is composed of 11 N/C machining centres, three AGVs, three conveyors, and a robot.

In this figure, a "part-flow deadlock" and an "AGV deadlock" situation can be easily found. The former occurs when a part on machine "J" intends to move to machine "K" and at the same time a part on K intends to move to J with each buffer full. The latter takes place when AGV1 has its routing as "Z3 → Z2 → Z1" and AGV2 as "Z1 → Z2 → Z3". In this section, the "AGV deadlock situation" is focused on. The subject of a "part-flow deadlock" may be found in the report by Kim and Tanchoco [5]

3.1 Deadlock avoidance algorithm

The dynamic resource-allocation policy decides how each AGV will request a resource, that is, a zone, which may affect the resource utilization and throughput. One extreme resource-allocation policy is the "request all policy". This policy reserves all of the zones to be used by the AGV within its whole routing in order that it can prevent deadlock. However, it causes extremely low resource utilization. Contrary to the request all policy, a simple and efficient resource allocation policy is proposed by Kim [7], which provides not only easy deadlock control but also high resource utilization to automated manufacturing systems.

The DAA in the case of two AGVs, which are operated in a 3×3 grid-type AGV layout as the above figure, is proposed as in the following pseudo-code. The DAA requests the AGV id and the current zone-requesting stage as arguments. The zone-requesting stage is the number of step since an AGV is entered into the system. Initially, an AGV demands the zone z_1 as well as z_2. Thereafter, it requests only the second zone $z_{current+2}$ from the current zone $z_{current}$ since the next zone $z_{current+1}$ has already been requested in the previous request. After checking the zone-requesting stage, the DAA requests the control system status information on whether or not the next zone $z_{current+1}$ is idle. If the zone $z_{current+1}$ is idle, the DAA moves to the deadlock detection module. Otherwise, it sends a "waiting" message to the AGV via the control system. The following provides brief details of the proposed DAA:

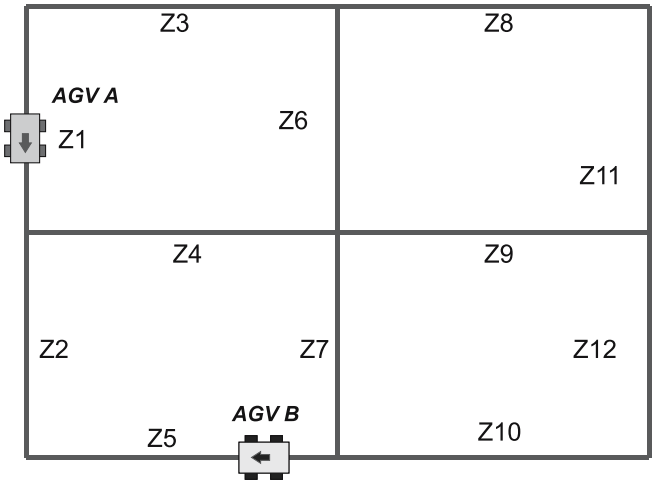


Fig. 6. A pilot FMS plant example

[Algorithm 1] DAA in the case of 2 AGVs

```
Deadlock Avoidance Algorithm (a_i, current)
{
if current == 0) then
   insert request edges a_i → z(i, current + 1) and
      a_i → z(i, current + 2)
else
   delete assignment edge z(i, current) → a_i
   insert request edge a_i → z(i, current + 2)
end if
calculate new path matrix
if (z(i, current + 1) is idle) then
   if (P[a_i, z(i, current + 1)] > 0) then
      if (a_i has alternative routings) then
         delete a_i → z_j which Φ(a_i, z_j) == 1 for all j
         GetNewRouting (a_i) //new routing
         insert request edges a_i → z(i, current + 1)
            and a_i → z(i, current + 2)
         calculate new path matrix
      else
         delete a'_i → z_j which Φ(a'_i, z_j) == 1 for all j
         GetNewRouting (a'_i) //new routing
         insert request edges a'_i → z(i, current + 1)
            and a'_i → z(i, current + 2)
         calculate new path matrix
      endif
   else
      change a'_i → z(i, current + 1) to z(i, current + 1) → a_i
   endif
else //z(i, current + 1) is busy
   wait //No action: not deadlock state but blocking state
endif
}
```

In the above algorithm, a_i and a'_i stand for the ith AGV and the other AGV, respectively. Zone z_j denotes the zone j. The zone identification number may be named at the layout designer's or the programmer's discretion. Φ is the graph representation matrix, which has a binary value. Therefore, $\Phi(a_i, z_j) = 1(0)$ means that there is (or is not) a directed arc from the AGV i to zone j. P is the path matrix, which has an integer value, $P(a_i, z_j) = n$ means that there are n paths from the AGV i to zone j. Lastly, the argument "*current*" is the current resource requesting stage number. "$current = 0$" means that the current stage is the initial resource requesting stage of a AGV, which is associated with the current event, i.e., the AGV is currently entered into the system. "$current = m$" means that the current stage

is the mth resource requesting stage, which is related with the current event. In other words, the AGV is located on the mth routing sequence.

3.2 Verification of the DAA in the case of 2 AGVs

The graph in Fig. 8 is called the resource-allocation graph (R-A graph). It has two columns (left column: the AGV column and right column: the zone column). In the R-A graph, the dotted line represents the newly added request edge. The double point line represents the newly added assignment edge, which is replaced by the request edge in the previous step. The normal line means there are no changes. Lastly, the check mark stands for the activated AGV. The "activated" AGV means that it has the priority in receiving commands from the control system.

Initially, AGV1 is located on Z1 and is destined for Z5, and AGV2 is on Z5 and is destined for Z4. Since Z1 and Z5 are already occupied by AGV1 and AGV2, respectively, Fig. 8a expresses this situation as the assignment edges (the normal line). In addition, both AGV1 and AGV2 are headed for Z2. Figure 8a

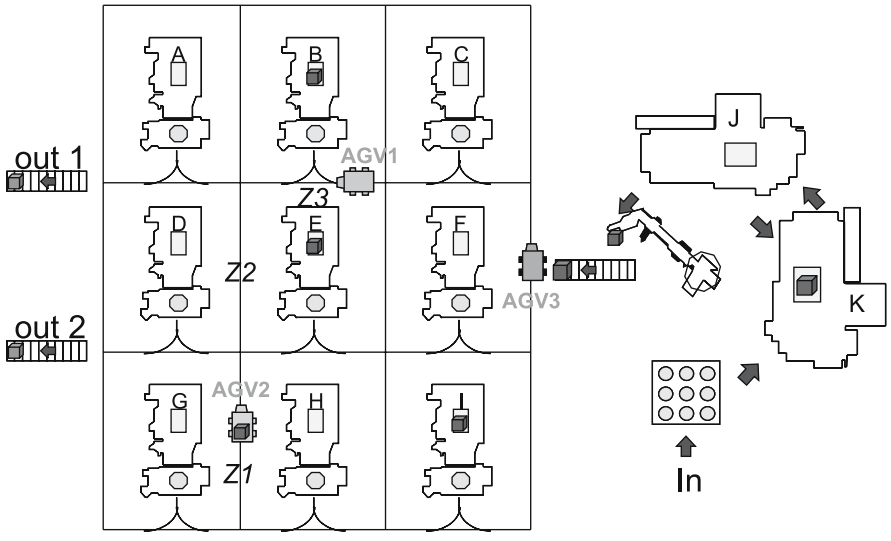


Fig. 7. A typical AGV system layout

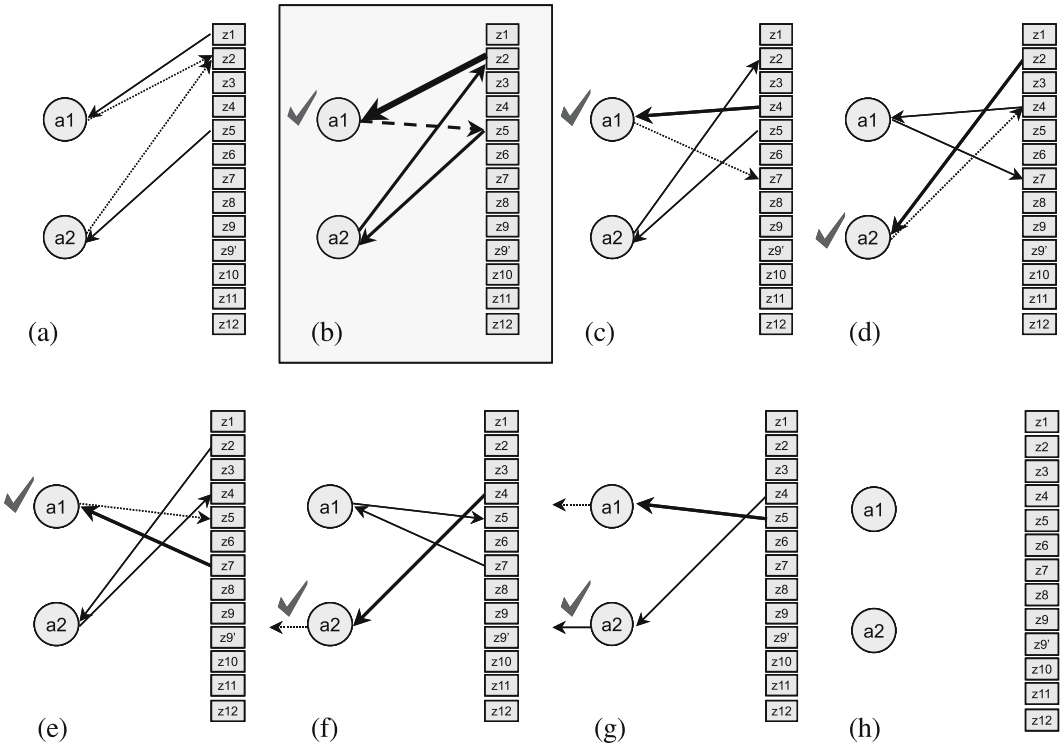


Fig. 8. An example in the case of two AGVs

expresses this situation as request edges (the dotted lines). Figure 8b represents the one-step advanced situation. AGV1 is activated by the check mark on the AGV1 node. However, the insertion of a request edge (AGV1 → Z5) makes the graph cyclic. In other words, it causes a deadlock situation, as shown in Fig. 8b. Therefore, an alternative routing for the AGV needs to be found. The rerouting function in the DAA searches an alternative routing, Z1 → Z4 → Z7 → Z5. The solution by the DAA is shown in Fig. 8c. AGV1 moves to Z4 (the double point line) not to Z2 and it is destined for Z7 (the dotted line). In these ways, the proposed deadlock avoidance procedure is performed. As a result, both AGV1 and AGV2 reach their own destinations safely, as shown in Fig. 8g.

3.3 Advantages to using the graph-theoretic approach

The advantages are classified as follows (1) the deadlock detection method and (2) the modeling advantage. The former is shown in Fig. 9b and the latter is represented in Fig. 9i,j.

Suppose that Z9 is allowed to travel in either direction with a double capacity, so that two AGVs can travel through it. If this layout is modeled using a Petri net, it will require too much time and effort to take Z9 into consideration. In particular, if we already have the Petri-net model, which considers Z9 with normal capacity, the Petri-net model needs to be totally redesigned in order to include the new property of Z9. However, the new method provides a very simple solution to include the new prop-

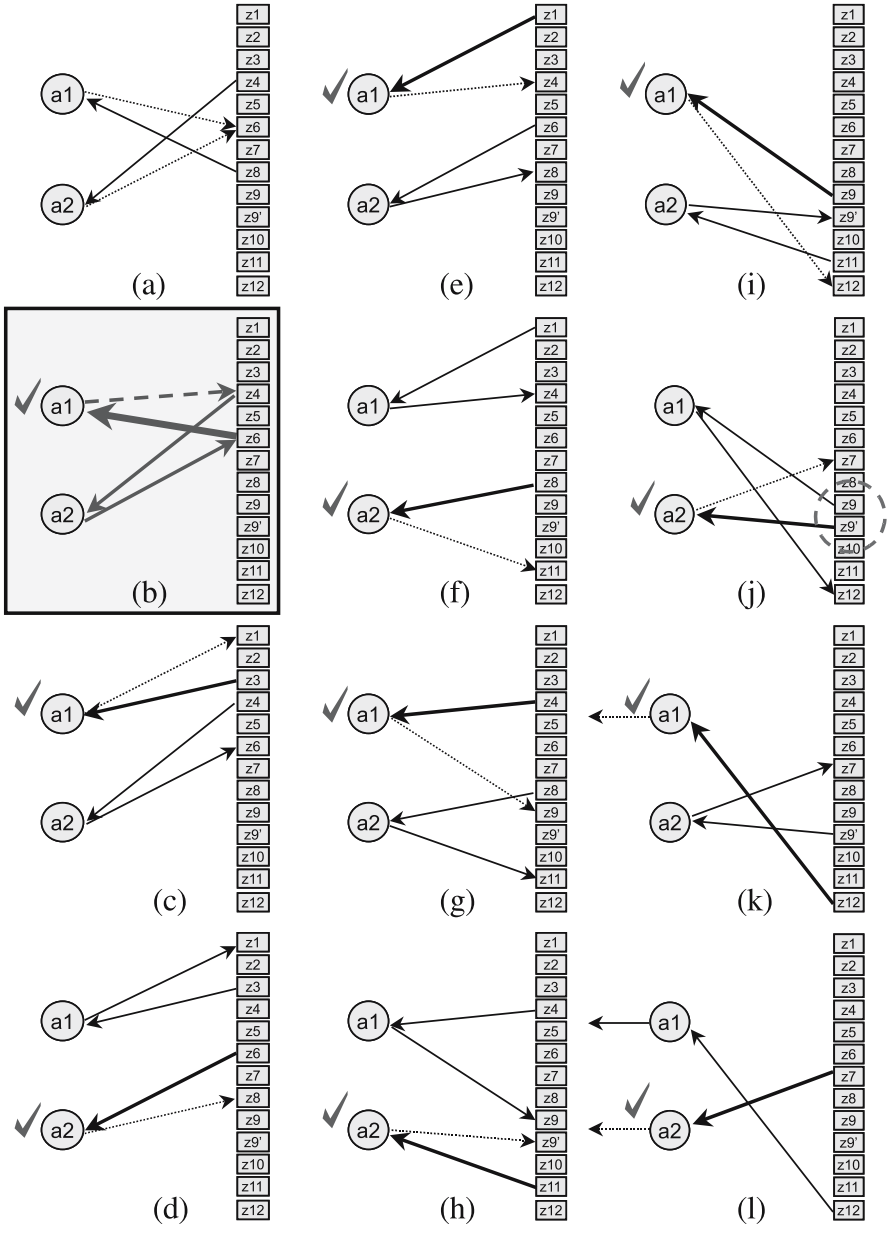


Fig. 9. An example showing the advantages of the proposed DAA

erty. Only another Z9 node (Z9′ here after) needs to be added to the existing model. As shown in Fig. 9i, adding the Z9′ node makes the graph acyclic, otherwise the graph would have a cycle. Furthermore, as shown in Fig. 9j, Zone 9 accommodates with AGVs without any conflict.

In summary, the example in Fig. 9 shows graphically both how the modified AGV layout can be easily modeled and how to control the layout to avoid a deadlock.

3.4 Data structure and flow chart for AGV control system

The AGV layout usually forms cross stripes or modified cross stripes, as shown in Fig. 10.

On these layouts, an AGV moves from the "START" position to the "DESTINATION" position, as the AGV controller commands. To control the AGVs without deadlock or conflict, it is necessary to know the information on the layout and the system status precisely. The following [Data structure 1] shows the data structure for the layout and system status in C language.

[Data structure 1] Data structure for layout and system status information

```
struct Matrix {
    int direction;
    int penalty;
    int next_direction;
} ZoneLayout[NO_ZONE+1][NO_ZONE+1];
int PathMatrix[NO_AGV+NO_ZONE+1][NO_AGV+NO_ZONE+1];
```

First of all, the zone layout is expressed as a structure template "ZoneLayout[i][j]". Its structure variable is made up of (1) "direction", (2) "penalty", and (3) "next_direction". Items (1) and (3) represent the direction of the AGV in the current zone and the next zone, respectively. In Fig. 10, it can be determined what the direction of the AGV means and how important it is. For example, if AGV A moves downward, the shortest path from the starting point (Z1) to the destination (Z3) is Z1 → Z4 → Z6 → Z3, whereas if AGV A is moved upward, the shortest path is Z1 → Z3. In this example, since AGV A moves downward, it is impossible for AGV A to have its routing as Z1 → Z3. Consequently, the direction of the AGV has a considerable effect on the AGV routing. In detail, the structure template "ZoneLayout[i][j]" means that an AGV moves from Z_i to Z_j. Accordingly, the structure variable "direction" and "next_direction" mean the direction of the AGV in Z_i and Z_j, respectively. The direction is symbolized by an integer variable. That is, "1" means the direction "→" and "↓", and "2" means the direction "←" and "↑". For example, "ZoneLayout[4][6]" has {1, 2, 2} as its structure variables. The first value in the brackets means that the direction of the current AGV in Z4 is toward "→", and the third value means that the direction of the AGV entering Z6 after being released in Z4 is "↑". The third value "penalty" refers to the cost of moving from Z4 to Z6.

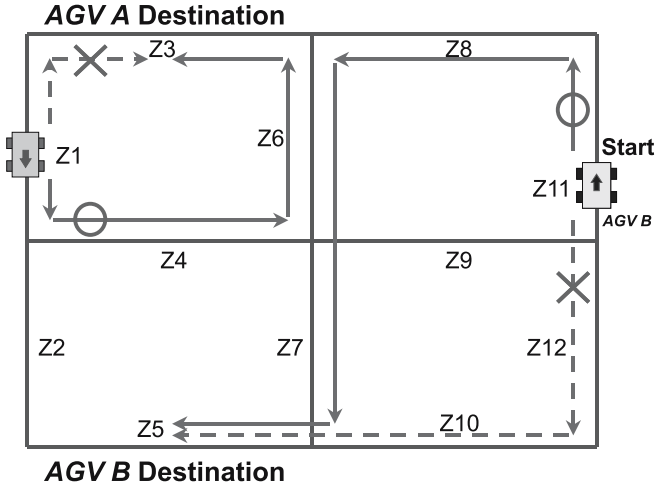


Fig. 10. Example showing the advantages of the proposed DAA

Secondly, consider the matrix representation, which represents the current system status information. A two dimensional array "int PathMatrix[i][j]" represents the number of paths connecting i and j, which was explained in Sect. 2.

Lastly, the two dimensional array "Boolean GraphRepresent[i][j]" reveals whether or not there is a link between i and j. Thus far, the proposed data structure related to the layout and system status information, and its usage has been explained.

An AGV needs to know the sequence in order to reach its destination from its starting point, that is, its routing information. First of all, structure template "JobList[n][m]" represents the mth job of the AGV n and its structure variables "From" and "To" mean the starting point and the destination, respectively. A job means a task, for instance, inventory replenishment, for which an AGV should move products from an unloading area (zone A) to the warehouse (zone B) in a retail or wholesale warehouse. Lastly, structure template "Routing_AGV[p][q]" represents the qth route of AGV p and its structure variables "zone" and "direction" mean the zone it passes through and the direction("1" or "2"), respectively.

[Data structure 2] Data structure for layout and system status information

```
struct Job {
    int From;
    int To;
} JobList[NO_AGV+1][NO_JOB+1];
struct Routing {
    int zone;
    int direction;
} Routing_AGV[NO_AGV+1][NO_ROUTE+1];
```

The AGV control system consists of several functions. These functions include the current system status information gathering function, feasible action generation function, the deadlock check function, and the action command function. Figure 11 shows what sequence these functions are executed in an AGV control system in order to avoid deadlocks and conflicts.

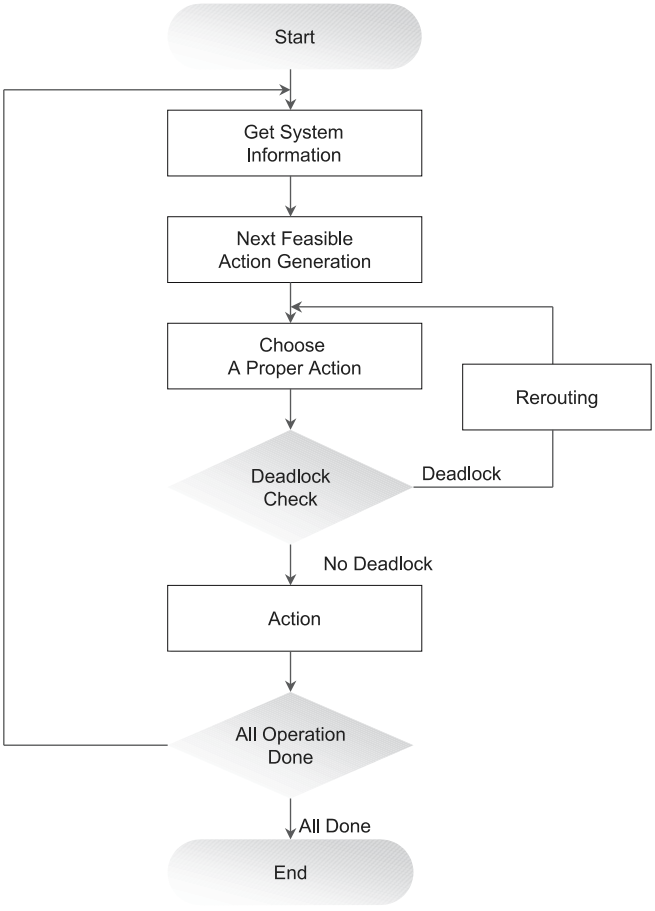


Fig. 11. The flow chart of AGV control system

4 Experiments

In this paper, two kinds of routing strategies (algorithms) are proposed: (1) the shortest path algorithm and (2) the load-based path algorithm. Load means the number of reservation on each zone by each job. In other words, as a zone is more likely to be occupied by AGVs, the zone has heavier load than other zones by the number of anticipated occupations. Each algorithm has its own characteristics. The former focuses on the total distance of the vehicle and the latter focuses on the load. The shortest path algorithm provides the shortest distance but it may result in a deadlock situation. The load-based path algorithm suggests the routing that takes the load into account. Hence, it provides a relatively long distance, but can prevent a deadlock situation. For example (see Fig. 12), suppose that AGV1 is headed for Z9 and AGV2 is headed for Z1.

In the case of applying the "shortest path algorithm", the routing of AGV1 from Z1 to Z9 is Z1 → Z4 → Z9 and that of AGV2 from Z9 → Z1 is Z9 → Z4 → Z1. Then AGV1 and AGV2 will definitely face the deadlock situation. On the other hand, in the case of applying the load-based algorithm, the routing of AGV1 from Z1 to Z9 is Z1 → Z2 → Z5 → Z7 → Z9.

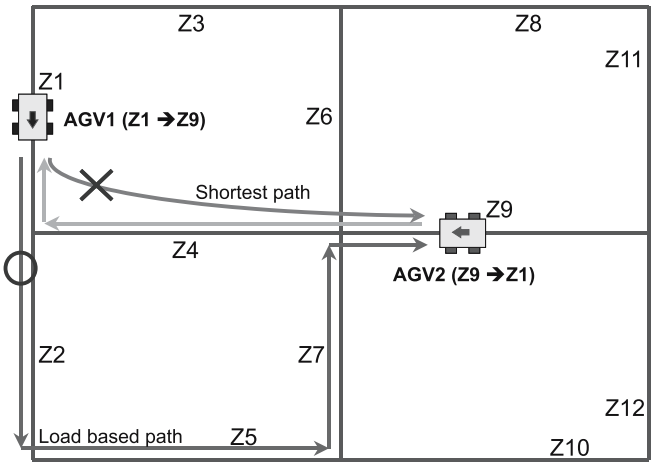


Fig. 12. Two different routing strategies

It will require a longer distance and time than the shortest path algorithm, but since a deadlock situation will not occur, the load-based algorithm is consequently more efficient.

In the following section, an experimental design and results are shown. Through these experiments, it will be verified that no deadlocks occur.

4.1 Experimental design

Simple experiments were performed to compare the two strategies. As shown in Fig. 12, the layout has a cross stripes form and bi-directional guide-paths. Two AGVs are on the layout and the DAA is applied for the two AGVs (see Sect. 3) in order to control the AGV system.

The AGV operational strategy is as follows. First, the AGV receives its own task (from the starting point to its destination, i.e., Z3 → Z12) and performs that task. While under operation, if the deadlock detection module finds that an AGV will face a deadlock situation in advance, the DAA takes actions (re-routing) to avoid the deadlock. Lastly, two steps are performed before its check point, and the AGV receives its next task (i.e., another destination or battery charging area). One hundred experiments were performed with different random numbers (seed). The random number was generated by using C code for the PMMLCG with $m = 2^{31} - 1$ and $a = 630, 360, 016$ based the report by Marse and Roberts [8]. Each experiment performs 20 jobs per AGV. An experiment ends when each AGV completes its particular jobs.

4.2 Experimental results

The algorithm tests were performed with 100 different seeds. Table 1 shows the results of the experiments.

The "No." columns show the trial number. The "S.P." columns show the number of deadlocks detected in advance, when the shortest path algorithm is applied to the routing strategy and the DAA to the control system. The "L.P." columns show the number

of deadlocks detected in advance, when the load-based path algorithm and the DAA are applied. The bottom row in Table 1 shows some helpful statistics in understanding the results. In terms of the total number of deadlocks avoided, the number of deadlocks in the case of S.P. (total number: 301) is more than that in the case of L.P. (total number: 276). The result reflects the better effectiveness of applying the load-based path algorithm.

In order to verify the effectiveness statistically, the paired-t confidence interval was obtained and the two-sample t-tests were performed. Let X_{1i} and X_{2i} be the observations of samples 1 and 2, respectively. The paired-t confidence interval is defined as follows [9] (n is the number of observations, i.e., the number of replications):

$$Z(n) = \frac{\sum_{i=1}^{n} Z_i}{n}, \quad \text{where } Z_i = X_{1i} - X_{2i} \text{ for } i = 1, 2, \ldots, n$$

$$\text{and } Var[Z(n)] = \frac{\sum_{i=1}^{n} [Z_i - Z(n)]^2}{n(n-1)}$$

and the $100(1-\alpha)\%$ confidence interval is

$$Z(n) \pm t_{n-1,1-\alpha/2}\sqrt{Var[Z(n)]}.$$

Thus, the confidence interval when $\alpha = 0.10$ and 0.132 is $[-0.503, 0.003]$, and $[-0.481, -0.019]$, respectively. As a result, although the confidence interval does not include the zero point (0) at the 0.10 level, it includes the zero point at the 0.132 level. In the final analysis, it is not significant, but there is non-neglibile evidence of a difference.

In addition, the two-sample t-tests were performed as follows [10]:

1. $H_0 : \mu_1 = \mu_2$ or $\mu_{diff} = \mu_1 - \mu_2 = 0$.
2. $H_1 : \mu_1 \neq \mu_2$ or $\mu_{diff} = \mu_1 - \mu_2 \neq 0$.
3. $\alpha = 0.10$.
4. Critical region: $t < -1.660$, and $t > 1.660$, where $t = \frac{Z(n)-0}{s_d/\sqrt{n}}$ with $v = 100$ degrees of freedom.
5. Computations: The sample mean and standard deviation for the Z_i's are $Z(n) = -0.2500$,

   $$\text{and } s_d = \frac{\sum_{i=1}^{n} [Z_i - Z(n)]^2}{(n-1)} = 1.5267.$$

   $$\text{Therefore, } t = \frac{Z(n)-0}{s_d/\sqrt{n}} = \frac{-0.25-0}{1.5267/\sqrt{100}} = 1.638.$$

6. Conclusions: The t-statistic is not significant at the 0.10 level. However, the p-value is $P = P(|T| \succ 1.638) \cong 0.105$. As a result, there is non-negligible evidence of a difference.

Table 1. The experimental result of two routing strategies

No.	S.P.	L.P.	No.	S.P.	L.P.	No.	S.P.	L.P.	No.	S.P.	L.P.	No.	S.P.	L.P.
1	2	4	21	3	1	41	3	1	61	4	4	81	3	3
2	3	4	22	3	2	42	2	1	62	1	1	82	2	1
3	7	3	23	3	3	43	4	4	63	5	5	83	3	3
4	1	2	24	5	5	44	4	3	64	2	2	84	2	2
5	2	4	25	3	2	45	1	2	65	1	0	85	3	7
6	5	3	26	1	1	46	3	1	66	4	4	86	2	2
7	5	4	27	4	3	47	2	5	67	2	3	87	1	2
8	1	1	28	4	5	48	4	3	68	2	0	88	2	1
9	5	3	29	2	2	49	3	1	69	4	4	89	3	2
10	5	1	30	4	5	50	3	3	70	2	3	90	4	2
11	5	2	31	2	5	51	1	1	71	4	4	91	3	2
12	2	4	32	3	3	52	1	4	72	3	3	92	3	5
13	0	4	33	2	1	53	1	1	73	3	2	93	2	2
14	5	4	34	2	1	54	3	3	74	1	2	94	4	4
15	3	5	35	3	3	55	2	3	75	3	2	95	3	2
16	6	4	36	1	2	56	3	2	76	7	7	96	4	4
17	1	0	37	3	3	57	3	3	77	6	7	97	4	2
18	4	3	38	2	0	58	2	2	78	3	3	98	3	3
19	4	3	39	3	2	59	5	6	79	3	0	99	5	6
20	5	1	40	2	1	60	2	1	80	3	2	100	6	4
Tot	301	276	μ	3.01	2.76	Var	2.50	2.03	Std	1.58	1.43			

5 Conclusions

5.1 Concluding remarks

With the rapid development of computer and automation technologies, most of today's manufacturing systems and material handling and distribution systems are evolving toward automated manufacturing systems (AMS). This is because they have the potential for providing a high level of performance, i.e., high productivity, reduction in lead time, and high adaptability to continuously changing demand. In this study, a deadlock avoidance algorithm was proposed in order to operate the whole AMS without deadlocks. Even though many researchers have presented many deadlock avoidance methods, almost all use additional devices such as detours, spurs, or buffers. The deadlock avoidance method proposed in this paper required no additional devices.

Many researchers devised modeling techniques for the sake of easy deadlock control, for example Petri nets. However, modeling an AMS using a Petri net is a time-consuming task. To make matters worse, whenever a small change in physical layout (structure) and/or a logical structure occurs, it takes too much time to revise the model. Revising the model causes system-wide changes in the existing model. In this paper, a simple and easily adaptable modeling technique and a DAA for AGV systems was applied. This technique can also be used in various areas, for example, manufacturing system, material handling and distribution system, and so on.

Lastly, a preferred AGV routing strategy is suggested, which was appropriate for the proposed DAA. As a result, the load-based path routing strategy was chosen, which shows better performance to the shortest path routing strategy.

5.2 Further research

In Sect. 3, only a basic idea for the DAA in the case of more than 2 AGVs was suggested. If the DAA is complete in the case of more than 2 AGVs, the AGV deadlock avoidance and collision-free control is fully accomplished. Thus, the remaining study, if any, involves improvements in the system performance, such as

reducing lead time and minimizing the total distance of the AGV. It would be desirable to develop a full set of shop-floor controllers that include a scheduler, a dispatcher, and a monitoring module as well as a deadlock controller.

References

1. Belik F (1990) An efficient deadlock avoidance technique. IEEE Trans Comput 39(7):882–888
2. Kundo S, Akyildiz IF (1989) Deadlock free buffer allocation inclosed queueing networks. Queueing Syst 4:47–56
3. Viswanadham N, Narahari Y, Johnson TL (1990) Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri-net models. IEEE Trans Robot Automat 6(6):713–723
4. Wysk RA, Yang BN, Joshi S (1994) Resolution of deadlocks in flexible manufacturing systems: avoidance and recovery approaches. J Manuf Syst 13(2):128–138
5. Kim CW, Tanchoco JMA (1991) Conflict-free shortest-time bidirectional AGV routing. Int J Prod Res 29:377–2391
6. Lee CC, Lin JT (1995) Deadlock prediction and avoidance based on Petri nets for zone-control automated guided vehicle systems. Int J Prod Res 33(12):3249–3265
7. Kim C-O, Kim SS (1996) An efficient real-time deadlock-free control algorithm for automated manufacturing systems. Int J Prod Res 35:1545–1560
8. Marse K, Roberts SD (1983) Implementing a portable FORTRAN uniform (0, 1) generator. Simulation 41:135–139
9. Law AM, Kelton WD (1991) Simulation modeling & analysis. McGraw-Hill
10. Walpole RE, Myers RH (1985) Probability and statistics for engineers and scientists. Maxwell Macmillan