2005 American Control Conference
June 8-10, 2005. Portland, OR, USA

Deadlock Avoidance Algorithm for Flexible Manufacturing Systems By Calculating Effective Free Space of Circuits

†Wenle Zhang and Robert P. Judd
School of Electrical Engineering and Computer Science
Ohio University
Athens, Ohio 45701

Abstract: Modern flexible manufacturing systems (FMS) are highly automated and flexible in which raw parts of various types are processed concurrently. Deadlock issue arises easily in these systems due to shared equipment usage and high production flexibility. This paper presents a deadlock avoidance algorithm for FMS with free choices in part routing by calculation of effective free space of circuits of the digraph model. The algorithm is highly permissive since the effective free space calculation captures more parts flow dynamics, especially when there exist multiple knots in the digraph model. It runs in polynomial time once the set of circuits is computed offline. Simulation results are provided.

Keywords: flexible manufacturing system, choice, digraph, circuit, deadlock avoidance.

1. Introduction

Modern flexible manufacturing systems (FMS) are highly automated and flexible in which raw parts of various types are processed concurrently. Deadlock issue arises easily in these systems due to shared equipment usage and high production flexibility. To increase equipment utilization and maximize productivity, it is crucial for a FMS to operate without deadlock. Research on deadlock detection, prevention and avoidance for flexible manufacturing systems has been rather active recently. Some of the significant works adopted Petri net (PN) models [1, 2, 4, 6, 10, 12, and 16] as a formalism to describe the manufacturing system. Another formalism is to describe the manufacturing system using graphs [3, 5, 8-9, 11, 13-15]. In this approach the vertices represent resources and the arcs (edges) represent product part flows between resources.

It is well known that it is difficult to detect impending deadlocks that are arbitrary steps away from primary deadlocks. Fanti [5] studied second level deadlock – the impending deadlock one step away from a primary deadlock. Barkaoui [2] used a one step look-ahead controller, which cannot avoid impending deadlocks that are more than one step away. Our previous work [14], in which no free choice is allowed in part routing, avoids deadlocks, especially impending deadlocks by dynamically evaluating the order of circuits.

The major contribution of this paper is the development of a new deadlock avoidance algorithm which extends our previous results [14] on deadlock avoidance for FMS's *without free choice* in part routing to systems *with free choices* in part routing. Because of choices introduced, part flow dynamics become much more complex, order evaluation method given by [14] is no longer valid due to the added routing flexibility. The extension is made possible by the systematic circuit analysis presented in Zhang [15], where concepts such as broken circuit, basic circuit, supremal circuit, were presented. These concepts help to decrease number of necessary circuits for deadlock checking thus increasing efficiency and permissiveness of our deadlock avoidance algorithm. The presented algorithm is unique in that it avoids both primary deadlocks and impending deadlocks that are arbitrary steps away from primary deadlocks. It can be shown that the presented algorithm runs in polynomial time once the set of necessary circuits of the digraph is computed offline.

2. The System Model and Deadlock

We consider a flexible manufacturing system that consists of a set R of a finite number of resources (such as, NC machines, robots, buffers, etc.) and a set P of a finite number of part types that the system can produce. Each resource $r \in R$ has a *capacity*, denoted as C_r, which can be considered as a multiple of identical units. The capacity can be naturally extended to a set of resources, $R_1 \subseteq R$, as C_{R1}. Each part type $p \in P$ is assigned a process plan that defines a finite number of steps of operations need to be performed on parts of the type. We assume that each step be performed on exactly one resource. Thus a process plan p can be represented as a sequence of resources $p=r_1\text{-}r_2\text{-}\ldots\text{-}r_m$. In case of a choice step, the next resource can be chosen from more than one resource. A choice step is indicated by a pair of parentheses in the process plan. Inside the pair of parentheses are the possible next resources separated by commas. A choice step can recursively contain choice steps. An example plan with choices is given in the following,

```
p1=r1-(r4-r6,r2-(r1,r5),r3-r5)-r4-(r5,r6)
   1   2  3  4   5  6   7  8    9  10 11
```

There are three choice steps in this plan. The first choice step is that after r_1 the part can go to any one of r_4, r_2 or r_3, thus leading to 3 choice branches which merge at second r_4. The second choice is that after r_2 the part goes to either r_1 or r_5. And the third choice is that after the second r_4 the part goes to either r_5 or r_6. The steps are sequentially numbered in the order they are listed. If a part is at step 7, then its next step is 8. If a part is at step 4, then its next step is 5 or 6.

Once the system is in operation, there will be a set Q of parts in the system at any given time. Each part $q \in Q$ belongs to a part type $p \in P$, denoted as $P_q=p$. Each part has a unique current step, denoted as S_q, which can be considered as the state of the part. The *state* of the system, denoted as $\boldsymbol{n}$, is defined as a vector of C_R elements corresponding to all parts currently in the system. An element of $\boldsymbol{n}$ has value 0 for an empty resource unit. The state changes with parts flowing through the system, such as loading a new part, unloading a finished part or transporting a part from one resource to the next resource. A part q that has exited the system or is still waiting for being loaded has $S_q = 0$.

† Corresponding author. Phone: 740-597-1481, Fax: 740-593-0007, Email: zhangw@bobcat.ent.ohiou.edu

For deadlock avoidance purpose, a flexible manufacturing system can be modeled by a directed graph, $G = (R, A)$, which is constructed from all process plans, where G consists of a set R of *vertices* and a set A of *directed arcs*. Each vertex represents a resource. A directed arc a is drawn from vertex r_1 to vertex r_2, if r_2 immediately follows r_1 (including choices) in at least one process plan, denoted as $a = r_1r_2$. So, a step has one or more arcs (called *choice arcs*), corresponding to the current resource (vertex), called *tail*, and one next resource for a non-choice step or more than one next resource for a choice step, called *head*(s).

A *subgraph* $G_1 = (R_1, A_1)$ of G consists of a subset of the vertices and a subset of arcs of G such that all the arcs in A_1 connect vertices in R_1. From graph theory, we know that a *path* is defined to be a sequence of vertices $r_0r_1r_2\ldots r_k$, and a *circuit* is a path with $r_0 = r_k$. A circuit is *simple* if it does not contain any other circuit. With choice introduced, we *generalize* a circuit as a sub-graph that is strongly connected.

A part q is *enabled* in state $\boldsymbol{n}$ if any of its next resource(s) is free. If part q is currently being processed in resource r_1 and the next free resource is r_2, then q enabled means that once r_1 finishes its operation on q, the part can be transported or moved from r_1 to r_2. A system state $\boldsymbol{n}$ is *live* if a sequence of part moves exists such that the system can be emptied. Otherwise, the state is in *deadlock*. Deadlocks can be further categorized into two major types, *primary deadlock* and *impending deadlock*. A system state $\boldsymbol{n}$ is in *primary deadlock* if a circular wait situation exists [1]. A primary deadlock can be understood as a circuit in the digraph model is filled with parts where no part is enabled. A system state $\boldsymbol{n}$ is in *impending deadlock* if parts exist in the system that can be moved; however, the system will inevitably enter a primary deadlock after a finite number of part moves.

3. Summary of Circuits Analysis

Given a system digraph, there are existing methods that find all simple circuits [7] [11]. In the following, we assume all simple circuits are given as a set C_S and discuss how to find the set of circuits sufficient for deadlock avoidance—called *necessary circuit set*, denoted as C.

Definition 3.1: If all choice arcs $a_1, a_2, \ldots, a_k$ of a choice step are on k different simple circuits, $c_1, c_2, \ldots, c_k$ of C_S, then the union circuit $c = c_1 \cup c_2 \cup \ldots \cup c_k$ is called a *choice circuit*.

So, each choice step has a corresponding choice circuit. A choice arc forms an escape path from a component circuit of the choice circuit for the part at the corresponding choice step. Such a circuit is called a broken circuit.

Definition 3.2: A circuit is broken if there is a resource (vertex) on the circuit where a choice step using the resource as a tail has at least one choice arc that is not on the circuit.

If a choice circuit has a choice arc (but not all) of another choice circuit, then it is broken. If the union circuit of the two (or more) choice circuits whose intersection has a choice arc from every choice circuit, then it is called *a multi-choice circuit*. A multi-choice circuit is non-broken if it contains all choice arcs from every component choice circuit.

Definition 3.3: A *basic circuit* is defined as a non-broken circuit that does not contain any other non-broken circuit.

Theorem 3.1: A broken circuit c that does not contain any basic circuit will not generate deadlock.
Proof: See [15]. ■

Then basic circuits are the smallest deadlock units of a system graph. Let C_B denote the set of all basic circuits of a system graph. All other circuits of the necessary circuit set C can then be formed by feasible unions of all basic circuits. Removing broken circuits and thus possible unions with them from the set C can reduce the size of C significantly.

The size of C can be further reduced. Given N (>1) different circuits all with the same set of vertices, $c_1 = (R_1, A_1)$, $c_2 = (R_1, A_2)$, …, and $c_N = (R_1, A_N)$, if $A_1 \subset A_k$, $A_2 \subset A_k$, …, $A_{k-1} \subset A_k$, $A_{k+1} \subset A_k$, …, $A_N \subset A_k$, then c_k is said to be the *supremal* circuit among the N circuits. Then c_k *covers* every other circuit. So, if given two circuits c_1, c_2 and c_2 covers c_1, then c_1 and c_2 have the same set of resources and c_2 has all arcs of c_1 and at least one more arc that c_1 does not have.

*Theorem*3.2: Given two circuits, c_1 and c_2, if c_2 covers c_1, then c_1 can be removed from the set C, that is, it does not need to be checked for deadlock avoidance.
Proof: See [15]. ■

The set of basic circuits C_B should then be updated by replacing a basic circuit (removed from C_B) with its supremal circuit if there is one.

Then, the set C of circuits should include all updated basic circuits in C_B and union circuits of basic circuits not covered by other circuit of C. And a circuit in C is either a basic circuit or a union circuit of two or more basic circuits.

Example 3.1: A manufacturing system has a digraph as shown in figure 3.1. The system makes two types of parts. The first type of parts has a process plan $p_1 = r_1$-(r_2, r_5)-r_3-r_4. The second type of parts has a process plan $p_2 = r_4$-r_3-r_2-r_1. It is easy to identify that the system graph consists of 4 simple circuits:

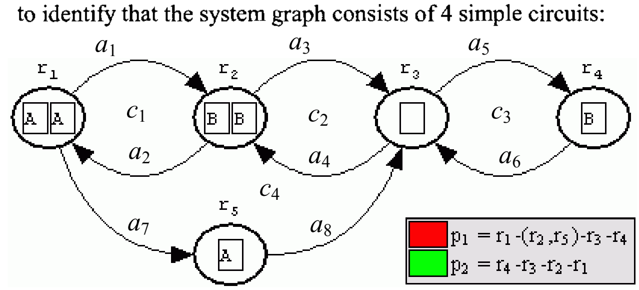

Figure 3.1 Basic circuits example

$c_1 = (\{r_1, r_2\}, \{a_1, a_2\})$, $c_2 = (\{r_2, r_3\}, \{a_3, a_4\})$
$c_3 = (\{r_3, r_4\}, \{a_5, a_6\})$, $c_4 = (\{r_1, r_2, r_3, r_5\}, \{a_2, a_4, a_7, a_8\})$

So, $C_S = \{c_1, c_2, c_3, c_4\}$. Arcs a_1 and a_7 are choice arcs of the choice step of p_1. Simple circuits c_1 and c_4 are both broken because of choice arcs a_1 and a_7, c_2 and c_3 are non-broken. The only choice circuit $c_5 = c_1 \cup c_4$ is non-broken. So, $C_B = \{c_2, c_3, c_5\}$. Possible unions are: $c_2 \cup c_3$, $c_2 \cup c_5$, $c_3 \cup c_5$, $c_2 \cup c_3 \cup c_5$; but $c_2 \cup c_5$ covers c_5 and $c_2 \cup c_3 \cup c_5$ covers $c_3 \cup c_5$. So, the updated $C_B = \{c_2, c_3, c_2 \cup c_5\}$ and $C = \{c_2, c_3, c_2 \cup c_3, c_2 \cup c_5, c_2 \cup c_3 \cup c_5\}$. If without removing the broken circuits and covered circuits, then C would contain 14 circuits.

4. Space Calculation of Circuits and Deadlock Avoidance Algorithm

In this section, we will first discuss calculation of effective free space of circuits and then presents the deadlock avoidance algorithm and property analysis.

4.1. Space Calculation of Circuits

With choice introduced, commitment can no longer be calculated based on arcs as in [13-14] because of existence of choice arcs. Instead, they will be calculated with respect to a circuit. If part q is at a choice step, then potentially q can move along any one of the choice arcs. However, q will move along only one of the choice arcs.

A unit of resource r_1 that is processing a part q is said to be *committed* to circuit c if q's next resource is r_2 and arc $a = r_1\text{–}r_2$ is on c, or if q is at a choice step, then all choice arcs are on c. We also say that q commits to circuit c. Let $M_{r,c,n}$ denote the number of units of resource r that are committed to circuit c when the system is in state $\boldsymbol{n}$.

Note that an empty unit is not committed and the total number of units of a resource committed can be less than the number of busy units. This happens when a busy unit is processing a part at its last step. This unit is also not committed. A unit of a resource that is on circuit c is *free* with respect to c if it is not committed to c.

The commitment can then be extended to the circuit $c = (R_1, A_1)$ as follows,

$$M_{c,n} = \Sigma\ M_{r,c,n},\ \text{for all } r \in R_1$$

Given the system in state $\boldsymbol{n}$, the *slack* of a circuit $c = (R_1, A_1)$, denoted as $K_{c,n}$, is defined as

$$K_{c,n} = C_{R1} - M_{c,n}$$

The slack can be understood as the number of available free resource units to allow for parts flow on the circuit.

Definition 4.1: Let $c_1, c_2, \ldots, c_m$, $m > 1$, be m component basic circuits of a circuit c of C. If $c_1 \cap c_2 \cap \ldots \cap c_m$ contains only a single capacity resource, then the resource is called a *knot* of c.

Definition 4.2: Given two component basic circuits c_1 and c_2 of a circuit c of C, with $k = c_1 \cap c_2$ being a knot. If in state $\boldsymbol{n}$, there exists a part in the system that will have to move through an arc of c_1 ending at k and will commit to an arc of c_2 starting at k, then c_1 is said to be *connected* to c_2 with respect to c, denote as $c_1 \rightarrow c_2$. If $c_1 \rightarrow c_2$ and $c_2 \rightarrow c_1$, then c_1 and c_2 are called *cross-connected*, denoted as $c_1 \leftrightarrow c_2$. If c has m component basic circuits $c_1, c_2, \ldots, c_m$, with $k = c_1 \cap c_2 \cap \ldots \cap c_m$ being a knot, then $c_1, c_2, \ldots, c_m$ are *cyclically connected* if $c_1 \rightarrow c_2, c_2 \rightarrow c_3, \ldots, c_m \rightarrow c_1$.

Definition 4.3: Given a circuit c of C with knot k. The order of knot k with respect to the circuit c in state $\boldsymbol{n}$, denoted as $O_{k,c,n}$, is defined as

$$O_{k,c,n} = \begin{cases} 1, & \text{if two or more basic circuits intersecting at } k \text{ are cyclically connected.} \\ 0, & \text{otherwise.} \end{cases}$$

Based on this definition, if c_1 and c_2, $c_2 \supset c_1$, are two circuits of C and $K=\{$all knots of $c_1\}$. Then,

$$O_{k,c2,n} = O_{k,c1,n},\ \forall\ k \in K.$$

The order definition can be extended to a circuit. Let c be a circuit that contains m knots, $k_1, k_2, \ldots, k_m$. Then, the order of c is given by

$$O_{c,n} = \sum_{i=1}^{m} O_{ki,c,n}$$

The order of a basic circuit is zero. Since a basic circuit either does not contain any other component basic circuits or has the intersection of the component basic circuits more than a vertex, so it has no knot. This is not the case in [14].

Definition 4.4: Let c be a circuit of C in state $\boldsymbol{n}$. The *effective free space* of c, denoted as $F_{c,n}$, is given by,

$$F_{c,n} = K_{c,n} - O_{c,n}$$

Theorem 4.1: Let G be the digraph of a flexible manufacturing system and C be the necessary circuit set of G. Then G is live in state $\boldsymbol{n}$ if $F_{c,n} > 0$, $\forall c \in C$.

Proof: This can be similarly proved as the corresponding theorem on systems without choice in [13]. ■

This theorem provides us with a sufficient condition for a system state to be live. Thus it can be used as the basis for developing our deadlock avoidance algorithm. It can be shown that connectedness of two circuits of C intersecting at a knot can be simplified to the connectedness of the two basic circuits intersecting at the knot.

Theorem 4.2: Let G be the digraph of a flexible manufacturing system and C be the necessary circuit set of G. Let c_1, c_2 be two circuits of C, and $c_1 \cap c_2 = k$ is a knot. Assume c_2 has m component basic circuits, $c_{1b}, c_{2b}, \ldots, c_{mb} \subseteq c_2$, $m > 0$, intersecting with c_1 at knot k. Then $c_1 \rightarrow c_2$ if and only if there exist i such that $c_1 \rightarrow c_{ib}$, $1 \leq i \leq m$.

Proof: If $c_1 \rightarrow c_2$, then there exists a part in the system that needs to enter c_2 through an arc of c_1 ending at k, since $c_{1b}, c_{2b}, \ldots, c_{mb}$ are the m basic circuits intersecting with c_1 at knot k, so the part has to enter one of them first, that is, $c_1 \rightarrow c_{ib}$. In the contrary, $c_1 \rightarrow c_{ib}$ implies $c_1 \rightarrow c_2$ since $c_{ib} \subseteq c_2$. ■

The significance of theorem 4.2 is that connectedness needs only be established among all basic circuits intersecting at a knot in order to calculate the order of the knot.

Example 4.1: Consider the system state given in figure 3.1, where all parts A are of type p_1 and all parts B are of type p_2. The only knot is resource r_3 and it is contained by the last three circuits of C. The connectedness among the three basic circuits can be determined as given in Table 4.1. Notice that c_2 is not connected to c_3, because part A does not have to move through c_2 into c_3; but $c_2 \cup c_5$ is connected to c_3. If two circuits do not intersect at a knot, table is labeled as N/A.

The commitment, order and space calculation for all circuits of C of the system graph is shown in the table 4.2.

Table 4.1 Connectedness table for example 4.1

Status	c_2	c_3	$c_2 \cup c_5$
c_2	N/A	No	N/A
c_3	Yes	N/A	Yes
$c_2 \cup c_5$	N/A	Yes	N/A

Table 4.2 Commitment, order and space for example 4.1

	c_2	c_3	$c_2 \cup c_3$	$c_2 \cup c_5$	$c_2 \cup c_3 \cup c_5$
M	0	1	1	5	6
O	0	0	0	0	1
F	3	1	3	1	0

According to theorem 4.1, the given system state is in deadlock since the last circuit has effective free space 0. And as a matter of fact, the state is actually an impending deadlock or second level deadlock.

Based on the sufficient conditions established above, a suite of algorithms, collectively called the deadlock avoidance algorithm, is developed for process control of actual manufacturing systems to detect and avoid deadlocks.

4.2. Calculation of Connectedness

First, the following presents a simple algorithm that determines the connectedness of two basic circuit c_1 and c_2 intersecting at knot k.

Algorithm 4.1: Connected(c_1, c_2, k)
Input: two basic circuit c_1, c_2 and knot k
Output: T – c_1 is connected to c_2, F – otherwise
Find R_1 on c_1 such that $\forall\ r_1 \in R_1$, arc r_1k is on c_1
Find R_2 on c_2 such that $\forall\ r_2 \in R_2$, arc $k\ r_2$ is on c_2
for each part q in Q **does**
  for each r_1 in R_1**do**
    for each r_2 in R_2 **do**
      if P_q has arc r_1kr_2 **then**
        if r_1 is after S_q in P_q **then return** T
    end for
  end for
end for
return F

Lemma 4.1: The complexity of algorithm 4.1 is in the order of $O(C_R^3|P_q|)$, where $|P_q|$ is the length of process plan P_q.

Proof: The algorithm first establishes arc r_1kr_2 by searching through circuit c_1 and c_2. Since c_1 and c_2 are basic, their length is limited by C_R, so this search can be done in linear time with order $O(C_R)$. Then the algorithm searches each part q in the system and the number of parts is again limited by the system capacity C_R. If part q's process plan P_q contains arc r_1kr_2 and if the arc is positioned after the part's current step S_q in its process plan, then according to definition 4.2 c_1 is connected to c_2. Searching arc r_1kr_2 in process plan P_q can be done with order $O(|P_q|)$, but it needs to search through both R_1 and R_2, that is at most C_R^2 (usually much smaller) times. The algorithm is smart in that once a part is found to have arc r_1kr_2 in its process plan after S_q, it returns true immediately. However, the worst case expense is still in the order of $O(C_R^3|P_q|)$. ∎

If there are m basic circuits in C, then the total number of pairs of basic circuits among m circuits is $1+2+\ldots+(m-1) = m(m-1)/2$. Connectedness for each pair needs to be established in order to apply theorem 4.1. If the connectedness among all pairs is updated before each part move, then it will be overly expensive. In order to improve the efficiency of our method, the incremental connectedness update will be considered. The idea is to first initialize the connectedness among all pairs to false, then updating only in two cases, i) upon loading a part into the system and ii) upon a part is moved into a knot k.

According to definition 4.2, case i) is the only way for two basic circuits to become connected. Case ii) is the only way for two basic circuits intersecting at k to become disconnected, assume no other part maintaining the connectedness.

The incremental update also needs a connectedness status table T that maintains status among all pairs of basic circuits. Note that each circuit might be connected to more than one other circuit. Presented in the following is the update algorithm.

Algorithm 4.2: UpdateConnected(K, q)
Input: K – set of knots of G, $|K| \le C_R$
    q – the part being loaded or moved into a knot
Output: updated status table T
for each knot k in K **do**
  let C_k be the set of basic circuits intersecting at k
  if P_q contains k **then**
    for each pair (c_1, c_2) in C_k **do**
      find R_1 on c_1 such that $\forall r_1 \in R_1$, arc r_1k is on c_1
      find R_2 on c_2 such that $\forall r_2 \in R_2$, arc $k\ r_2$ is on c_2
      for each r_1 in R_1**do**
        for each r_2 in R_2 **do**
          if P_q contains arc r_1kr_2 **then** $T(c_1, c_2)$ = T
        end for
      end for
    end for
end for

Lemma 4.2: The complexity of algorithm 4.2 is in the order of $O(m^2C_R^4|P_q|)$, where m is the number of basic circuits in C.

Proof: The outer for loop needs to repeat at most C_R times. The first inner for loop needs to repeat at most $m(m-1)/2$. According lemma 4.1, the cost for the two inner most for loops is in the order of $O(C_R^3|P_q|)$. So, algorithm 4.2 is in the order of $O(m^2C_R^4|P_q|)$. ∎

4.3. Calculation of Effective Free Space

Order of a knot k with respect to a non-basic circuit c can be calculated by first determining the cyclic connectedness among basic circuits of c intersecting at k. Let C_k be the set of basic circuits intersecting at k. Then, with the connectedness status table T, cyclic connectedness among circuits of C_k can be checked as described in the following:

Let $C_{con} = \{c_1, c_2, \ldots, c_n\}$ be a subset of connected circuits of C_k, that is $c_1 \to c_2 \to \ldots \to c_n$, $n > 1$. If c_n is not connected to any circuit of C_k, then circuits in C_{con} are not cyclically connected and should be removed from C_k. If $c_n \to c_{n+1} \in C_k$, $c_{n+1} \in C_{con}$ and $c_{n+1} \ne c_n$, then at least two circuits around k are cyclically connected; if $c_n \to c_{n+1} \in C_k$, but $c_{n+1} \notin C_{con}$, then add c_{n+1} to C_{con}. Repeat this process with the remaining circuits in C_k or the updated C_{con} until cyclic connectedness is found or C_k becomes empty. In the later case, no circuit is cyclically connected.

According to definition 4.3, the order of k with respect to circuit c is one if circuits of C_k are found cyclically connected, zero if no circuit is cyclically connected. The following algorithm implements the cyclic connectedness check and calculates the order.

Algorithm 4.3: KnotOrder(k, C_k, T)
Input: k – knot
    C_k – set of basic circuits intersecting at k
    T – the connectedness table
Output: 1 – two or more circuits of C_k are cyclically connected,
    0 – no circuit is cyclically connected
$C_{con} = \{\}$
while C_k is not empty **do**
  if C_{con} is empty **then** C_{con} = {first element of C_k}
  c_0 = last element of C_{con}
  connected = F
  for each c in C_k **do**
    if $T(c_0, c)$ equals T **then**
      connected = T, $c_0 = c$
      if c is in C_{con} **then return** 1

else add c to C_{con}
end for
if connected equals F **then**
remove C_{con} from C_k, $C_{con} = \{\}$
end while
return 0

With order calculated for a knot, the order calculation for a circuit is simple, which is presented as algorithm 4.4 as follows.

Algorithm 4.4: CircuitOrder(c, T)
Input: c – the circuit
T – the connectedness table
Output: O – order of circuit c
Let K be the set of knots of c, $|K|$ C_R
$O = 0$
for each knot k in K **do**
let C_k be the set of basic circuits intersecting at k
$O = O +$ KnotOrder(k, C_k, T)
end for

Lemma 4.3: The complexity of algorithm 4.4 is in the order of $O(m^2C_R)$.

Proof: If there are m basic circuits in C, then the number of basic circuits of C_k is limited by m. So the cost of algorithm 4.3 is in the order of $O(m^2)$ and thus that of algorithm 4.4 is in the order of $O(m^2C_R)$. ∎

Finally, the effective free space of a circuit can be easily calculated by definition 4.4. First, the slack of a circuit still needs to be calculated. To calculate the slack of a circuit c, it needs to go through the list of resources of the circuit, for each resource, set the slack to the resource capacity and subtract the commitment from the slack. The commitment of each resource r can be determined by checking each part q contained in r to see if all of part q's next step resources are on c or not.

Algorithm 4.5: CircuitEFSpace(c, T)
Input: c – the circuit whose space to be calculated
T – the connectedness table
Output: F_c– effective free space of circuit c
$K_c = 0$ // – slack of circuit c
for each resource r in R_c **do**
$K_c = K_c + C_r$ // C_r – capacity of r
for each part q in r **do**
let R_n be the set of q's next step resources
if R_n R_c **then**
$K_c = K_c - 1$ // q commits to an arc on c
end for
end for
$F_c = K_c -$ CircuitOrder(c, T)

Lemma 4.4: The complexity of algorithm 4.5 is in the order of $O(m^2C_R+C_R^2)$.

Proof: The first part of algorithm 4.5 calculates the slack of the circuit, which goes through each resource of the circuit and the number is limited by C_R. Within the for loop, each part is checked for whether its next resources are all on the circuit, which is also limited by C_R. Since algorithm 4.4 is in the order of $O(m^2C_R)$ by lemma 4.3, this algorithm is in the order of $O(m^2C_R+C_R^2)$. ∎

4.4. Deadlock Avoidance Algorithm

The algorithm first determines whether it needs to update the connectedness table or not. From above analysis, connectedness of circuits needs to be updated only if a part is loaded or moved into a knot. Then, based on theorem 4.1, the algorithm goes through each circuit of C to calculate the effective free space before each part is physically moved or loaded. As long as one circuit is found to have zero free space, the part move should be rejected, so the algorithm returns F, it returns T otherwise.

Algorithm 4.6: DAA(C, q, r)
Input: C – set of necessary circuits in digraph G of the system
q – part to be moved/loaded
r – q's next step resource (the requested move)
Output: T – accept move/load, F – reject
Let K be the set of knots of G
Let S_q be q's next step
if S_q equals 1 **then** // LOAD
$T =$ UpdateConnected(K, q) // update table T
if r is in K **then** // MOVE
$T(c_1, c_2) =$ Connected(c_1, c_2, r) // update table T
for each circuit c in C **do**
if CircuitEFSpace(c, T) equals zero **then return** F
end for
return T

Theorem 4.3: The deadlock avoidance algorithm 4.6 has a polynomial complexity.

Proof: If there are m basic circuits in C, then the cost of algorithm 4.6 is mainly in the effective free space calculation part. Calculating the effective free space of each circuit is in the order of $O(C_R^2 + C_Rm^2)$ given by lemma 4.4. But that needs to be done for every circuit of C, so overall it is in the order of $O(|C|(C_R^2 + C_Rm^2))$. The UpdateConnected (algorithm 4.2) is executed only when a part is first loaded or moved into a knot, which has a cost $O(m^2C_R^4|P_q|)$ from lemma 4.2. The call to Connected (algorithm 4.1) can be omitted. Still, the worst case complexity has an order of $O(|C|(C_R^2 + C_Rm^2)) + O(m^2C_R^4|P_q|)$. ∎

In order to avoid deadlock in the operation of a flexible manufacturing system, the above deadlock avoidance algorithm should be executed every time a new part is loaded or an existing part is moved. If the algorithm returns a true, the part load or move request can be granted, otherwise it should be denied.

5. Application Simulation

In order to show the effectiveness of the proposed deadlock avoidance algorithm, simulation has been run to calculate the state space allowed by the deadlock avoidance method on several examples. Simulation results show that the deadlock avoidance method is indeed correct.

Example 5.1: Consider the manufacturing cell shown in figure 5.1 (A case study in [4]). The cell is composed of three robots (R1, R2 and R3; each one can hold one product at a time) and four machines (M1, M2, M3 and M4; each one can process two products at a time). There are three loading buffers (named I1, I2 and I3) and three unloading buffers (named O1, O2 and O3) for loading and unloading the cell. The action area for robot R1 is I1, O3, M1, M3); for robot R2 is I2, O2, M1, M2, M3, M4; and for robot R3 is I3, O1, M2, M4.

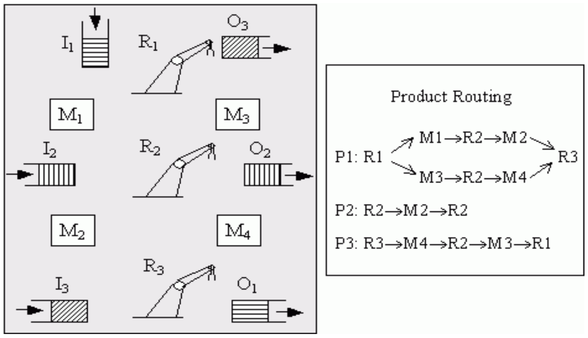

Figure 5.1 The manufacturing cell for example 5.1

The cell manufactures three types of products P1, P2 and P3, with routing given in figure 5.1. The directed graph is shown is figure 5.2, where the simple circuits identified are labeled.

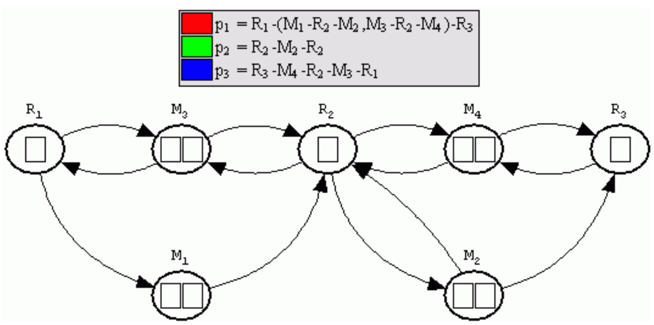

Figure 5.2 Digraph for example 5.1

Due to the choice step at R_1, both simple circuit c_1 and c_6 are broken. The choice circuit is c_1 c_6 which is covered by c_8 = c_1 c_6 c_2. So the set of basic circuits is $C_B = \{c_2, c_3, c_4, c_5, c_7, c_8\}$. Basic circuit c_7 is covered by its supremal circuit c_3 c_4 c_5 c_7. After removing circuits covered by their corresponding supremal circuits, the circuits set C is found to be, $C = \{ c_2, c_3, c_4, c_5, c_8, c_2\ c_3, c_2\ c_5, c_3\ c_4, c_3\ c_5, c_8\ c_3, c_8\ c_5, c_2\ c_3\ c_4, c_2\ c_3\ c_5, c_8\ c_3\ c_4, c_8\ c_3\ c_5, c_3\ c_4\ c_5\ c_7, c_2\ c_3\ c_4\ c_5\ c_7, c_3\ c_4\ c_5\ c_7\ c_8\}$.

Simulation with the deadlock avoidance algorithm applied shows that 20801 live states are allowed out of total 22019 live states. That corresponds to a permissiveness as high as 20801/22019 = 94.5%.

7. Conclusions

A highly permissive, correct and polynomial complexity deadlock avoidance algorithm for flexible manufacturing systems with choices in part routing, which avoids both primary deadlocks and impending deadlocks that are arbitrary steps away from primary deadlocks, is presented. The algorithm achieves high permissiveness based on the dynamic effective free space calculation of circuits in the digraph model, which captures more parts flow dynamics and therefore avoids impending deadlocks – a type of deadlock more difficult to detect. As shown in the simulation section, the average percentage permissiveness is consistently above 90% among all the tested examples. However, the algorithm is based on the sufficient condition for a state to be live; the necessary condition has not yet been established that may contribute to even higher permissiveness. Developing the necessary condition will be one of our future research topics.

8. References

[1] Banaszak, Z. and B. Krogh, "Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows," IEEE Trans. on Rob. and Auto., vol. 6, no. 6, 1990, pp. 724-733.

[2] Barkaoui, K., and I. B. Abdallah, "Deadlock Avoidance in FMS Based on Structural Theory of Petri Nets," IEEE Symposium On Emerging Technologies and Factory Automation, V. 2, pp. 499-510, 1995.

[3] Cho, H., T.K. Kumaran, and R. Wysk, "Graph-Theoretic Deadlock Detection and Resolution for Flexible Manufacturing Systems," *IEEE Trans. on Rob. and Auto.*, vol. 11, no. 3, pp. 550-527.

[4] Ezpeleta, J., J. M. Colom, and J. Martinez, "A Petri Net Based Deadlock Prevention Policy for Flexible Manufacturing Systems," *IEEE Trans. on Rob. and Auto.*, V. 11, N. 2, pp. 173-184, April 1995.

[5] Fanti, M.P., Maione, B., Mascolo S., and Turchiano, B., "Event-Based Feedback Control for Deadlock Avoidance in Flexible Production Systems", *IEEE Trans. on Rob. and Auto.*, Vol. 13, no. 6, 1997, pp. 347-363.

[6] Hsieh, F. and S. Chang, "Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems," *IEEE Trans. Rob. and Auto.*, vol. 10, no. 2, 1994, pp. 196-209.

[7] Johnson, D. B., "Finding All The Elementary Circuits Of A Directed Graph", SIAM J. of Computing, Vol. 4, No. 1, 1975, pp. 77-84.

[8] Judd, R. P. and T. Faiz, "Deadlock Detection and Avoidance for a Class of Manufacturing Systems," *Proceedings of the 1995 American Control Conference*, pp. 3637-3641.

[9] Lawley, M. A., "Deadlock Avoidance for Production Systems with Flexible Routing", *IEEE Trans. on Rob. and Auto.*, Vol. 15, No. 3, 1999, pp. 497-509.

[10] Viswanadham, N., Y. Narahari, and T. Johnson, "Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models," *IEEE Trans. on Rob. and Auto.*, vol. 6, no. 6, 1990, pp. 713-723.

[11] Wysk, R., N. Yang and S. Joshi, "Detection of Deadlocks in Flexible Manufacturing Cells", *IEEE Trans. on Rob. and Auto.*, Vol.7, No.6, 1991, pp.853-859.

[12] Xing, K., B. Hu and H. Chen, "Deadlock avoidance policy for Petri-net modeling of flexible manufacturing systems with shared resources," *IEEE Trans. on Automatic Control.*, vol. 41, no. 2, 1996, pp. 289-295.

[13] Zhang, W, R. P. Judd and P. Deering, "Necessary And Sufficient Conditions For Deadlocks In Flexible Manufacturing Systems Based On A Digraph Model", *Asian Journal of Control*, Vol. 6, No. 2, 2004.

[14] Zhang, W., R. P. Judd and P. Paul, "Evaluating Order Of Circuits For Deadlock Avoidance In A Flexible Manufacturing System", *Proceedings of the 2003 American Control Conference*, pp. 3679-3683, June 2003, Denver.

[15] Zhang, W. and R. P. Judd, "Deadlock Avoidance For Flexible Manufacturing Systems With Choices Based On Digraph Circuit Analysis", *Proceedings of the 2004 American Control Conference*, pp. 3333-3338, Jun 29-Jul 2, 2004, Boston.

[16] Zhou, M. and F. DiCesare, "Parallel and Sequential Mutual Exclusion for Petri Net Modeling of Manufacturing Systems with Shared Resources," *IEEE Trans. on Rob and Auto.*, vol. 7, no. 4, 1992, pp. 550-527.