18

Robust Control for Single Unit Resource Allocation Systems

Shengyong Wang, Song Foh Chew and Mark Lawley
University of Akron, Southern Illinois University Edwardsville, and Purdue University
USA

1. Introduction

Supervisory control for deadlock-free resource allocation has been an active area of manufacturing systems research. Most work, however, assumes that allocated resources do not fail. Little research has addressed allocating resources that may fail. Automated manufacturing systems have many types of components that may fail unexpectedly. We develop robust controllers for single unit resource allocation systems with unreliable resources (Chew et al., 2008; Chew et al., 2011; Chew & Lawley, 2006; Lawley, 2002; Lawley & Sulistyono, 2002; Wang et al., 2008; Wang et al., 2009). These controllers guarantee that when unreliable resources fail, parts requiring failed resources do not block the production of parts not requiring failed resources. Further, while resources are down, the system is controlled so that when repair events occur, the system is in a safe and admissible state.

There is little manufacturing research literature on robust supervision. Reveliotis (1999) considers the case where parts requiring a failed resource can be re-routed or removed from the system through human intervention. Park & Lim (1999) address existence questions for robust supervisors. Hsieh (2004) develops methods that determine the feasibility of production given a set of resource failures modelled as the extraction of tokens from a Petri net. In contrast, our work models the failure of the workstation server while assuming that buffer space remains accessible after the failure event. We assume that when the server of a workstation fails, we can continue allocating its buffer space up to capacity, but that none of the waiting parts can be processed and thus cannot proceed along their routes until the server is repaired. We further assume that server failure does not prevent finished parts occupying the workstation's buffer space from being moved away from the workstation and proceeding along their routes. Finally, we assume that server failure does not damage or destroy the part being processed and that failure can only occur when the server is working. The last two assumptions are made for notational efficiency and presentation clarity. They can be easily relaxed by adding appropriate events and state variables to our treatment.

Our objective is to control the system so that failure of an unreliable resource does not prevent processing of parts not requiring the failed resource. When a resource fails, all parts in the system requiring the failed resource for future processing are unable to complete until the failed resource is repaired. Because these parts occupy buffer space, they can block production of parts not requiring the failed resource. Thus, we want to assure that, when unreliable resources fail, the buffer space allocation can evolve under normal operation so

that parts not requiring failed resources can continue production. Operation must continue to obey part routings and must assure that when a failed resource is repaired, the system is not in an unsafe state. We refer to supervisors guaranteeing this as *robust*.

The remainder of the chapter comprises the following sections. Most briefly, Section 2 discusses the way we model our systems. An example system is presented in this section to motivate properties that robust controllers must possess. In Section 3, we develop robust controllers for systems with multiple unreliable resources where each part type requires at most one unreliable resource. Specifically, Subsection 3.1 develops two robust controllers using a neighbourhood policy, a modified version of banker's algorithm, and a single step look ahead policy. Subsection 3.2 uses a resource order policy to construct another robust controller; Subsection 3.3 employs a notion of shared buffer capacity to develop a robust controller. Relaxing the restriction, Section 4 builds robust controllers for systems for which part types may require multiple unreliable resources. Finally, Section 5 concludes the chapter and discusses future research directions.

2. Modelling of robust control

There are two subsections in this section. Specifically, we will discuss the way we model our systems in Subsection 2.1. Subsection 2.2 will provide examples to motivate properties that robust controllers must possess.

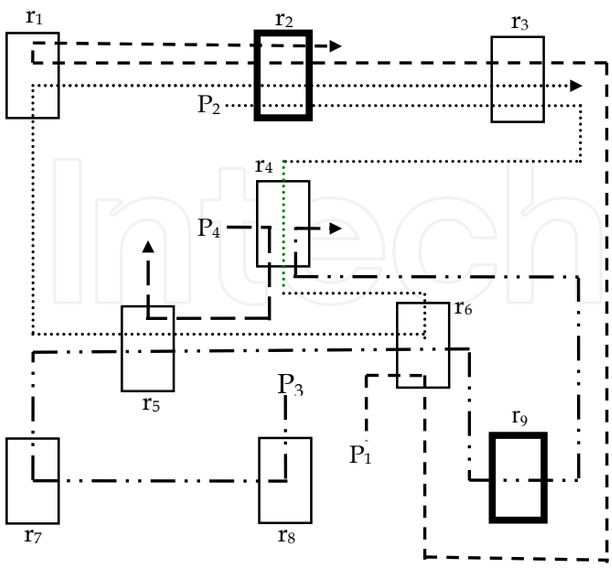
2.1 The discrete event system

We model our systems using the approach of Ramadge & Wonham (1987). This is necessary to define the properties that we want our supervisors to enforce. The following model is similar to that developed by Lawley & Sulistyono (2002), but differs in that now we have more complex failure scenarios and thus some of the underlying formalism has to be generalized. Figure 1 provides an example for the following development.

The system is defined as a 9–tuple vector $S = \langle R,C,P,\rho,Q,Q_0,\Sigma,\xi,\delta\rangle$. In S, R is the set of system resource types, with $R=R^R \cup R^U$, $R^R \cap R^U=\varnothing$, where R^R is the set of reliable resource types, not subject to failure and R^U is the set of unreliable resource types, subject to failure. Let $C=\langle C_i : i=1\ldots|R|\rangle$ where C_i is the capacity of the buffer space associated with system resource type $r_i \in R$.

The set P of part types is produced by the system with each part type $P_j \in P$ representing an ordered set of processing stages, $P_j=\langle P_{j1}\ldots P_{j|Pj|}\rangle$, where P_{jk} represents the k^{th} processing stage of P_j. Also, let $RP_{jk}=\langle P_{jk}\ldots P_{j|Pj|}\rangle$ be the *residual* part stages. We will use p_{jk} to represent a part instance of P_{jk}. Let $\rho:P_j \rightarrow R$ such that $\rho(P_{jk})$ returns the resource type required by P_{jk}. Thus, the route of P_j is $T_j=\langle\rho(P_{j1})\ldots\rho(P_{j|Pj|})\rangle$, and the residual route $RT_{jk}=\langle\rho(P_{jk})\ldots\rho(P_{j|Pj|})\rangle$. Finally, let $\Omega_i=\{P_{jk}:\rho(P_{jk})=r_i \in R\}$, the set of part type stages associated with resource $r_i \in R$.

We will suppose that our resource types are workstations with buffer space for staging and storing parts and a processor or server for operating on parts. We will use the standard assumption from queuing theory that the server is not idle so long as there are unfinished parts in a workstation's buffer space. The resource units that we are concerned with allocating are instances of the workstation's buffer space. The controllers that we design are not intended to allocate the server among parts waiting at the workstation. We assume this to be done by some local queuing discipline.



$R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$
$R^R = \{r_1, r_3, r_4, r_5, r_6, r_7, r_8\}$
$R^U = \{r_2, r_9\}$
$C = \langle C_1,C_2,C_3,C_4,C_5,C_6,C_7,C_8,C_9\rangle$
$= \langle 1,1,1,2,1,1,1,1,1\rangle$
$P = \{P_1, P_2, P_3, P_4\}$
$P_1 = \{P_{11}, P_{12}, P_{13}, P_{14}, P_{15}\}$
$T_1 = \langle r_6, r_3, r_2, r_1, r_2\rangle$
$P_2 = \{P_{21}, P_{22}, P_{23}, P_{24}, P_{25}, P_{26}, P_{27}, P_{28}\}$
$T_2 = \langle r_2, r_3, r_4, r_6, r_5, r_1, r_2, r_3\rangle$
$P_3 = \{P_{31}, P_{32}, P_{33}, P_{34}, P_{35}, P_{36}\}$
$T_3 = \langle r_8, r_7, r_5, r_6, r_9, r_4\rangle$
$P_4 = \{P_{41}, P_{42}\}$
$T_4 = \langle r_4, r_5\rangle$
$\Sigma_c = \{\alpha_{1i} : i=1...6\}\cup\{\alpha_{2i} : i=1...9\}\cup \{\alpha_{3i} : i=1...7\}\cup\{\alpha_{4i} : i=1...3\}$
$\Sigma_{u1} = \{\beta_{1i} : i=1...5\}\cup\{\beta_{2i} : i=1...8\}\cup \{\beta_{3i} : i=1...6\}\cup\{\beta_{4i} : i=1,2\}$
$\Sigma_{u2} = \{\kappa_2,\kappa_9,\eta_2,\eta_9\}$
$\psi^{\{r2\}} = \{\alpha_{1i} : i=1...6\}\cup\{\alpha_{2i} : i=1...9\}\cup \{\beta_{1i} : i=1...5\}\cup\{\beta_{2i} : i=1...8\}$
$\psi^{\{r9\}} = \{\alpha_{3i} : i=1...7\}\cup\{\beta_{3i} : i=1...6\}$
$\psi^{\{r2,r9\}} = \psi^{\{r2\}} \cup \psi^{\{r9\}}$

Fig. 1. An example system with two unreliable resources

Workstation *failure* will imply the failure of the workstation's server, not any of its buffer space. We will assume that when the server of a workstation fails, we can continue to allocate its buffer space up to capacity, but that none of the waiting parts can be processed and thus cannot proceed along their respective routes until the server is repaired. We further assume that server failure does not prevent finished parts occupying the workstation's buffer space from being moved away from the workstation and proceeding along their respective routes. Finally, we assume that server failure does not damage or destroy the part being processed and that failure can only occur when the server is working.

We are now in a position to define the system states and events. Let Q represent the set of system states, where $Q \ni q = \langle sv_i, y_{jk}, x_{jk} : i=1\ldots|R|, j=1\ldots|P| \text{ and } k=1\ldots|P_j|\rangle$, with sv_i being the status of the server of workstation i (0 if failed, 1 if operational), y_{jk} being the number of unfinished units of P_{jk} (parts waiting or in-process) located in the buffer space of $\rho(P_{jk})$, and x_{jk} being the number of finished units of P_{jk} located in the buffer space of $\rho(P_{jk})$. Q_0 is the set of initial states with $q_0 \in Q_0$ being the state in which no resources are allocated and all servers are operational. The dimension of q is $\sum_{j=1}^{|P|} 2|P_j|+|R|$.

Let $\Sigma=\Sigma_c\cup\Sigma_u$, where $\Sigma_c=\{\alpha_{jk} : j=1\ldots|P| \text{ and } k=1\ldots|P_j|+1\}$ is the set of controllable events with α_{jk} representing the allocation of $\rho(P_{jk})$ to a part instance of P_{jk}; that is, α_{jk} is the event that a part instance of a part type P_j advances into the buffer space of a workstation that will perform its k^{th} operation. Then, $\alpha_{j,|Pj|+1}$ represents a finished part of type P_j leaving the system. We assume that the supervisor controls the occurrences of these events through resource allocation decisions.

We have $\Sigma_u=\Sigma_{u1}\cup\Sigma_{u2}$ being the set of uncontrollable events where $\Sigma_{u1}=\{\beta_{jk} : j=1\ldots|P| \text{ and } k=1\ldots|P_j|\}$ represents the completion of service for P_{jk}. Then, $\Sigma_{u2}=\{\kappa_i,\eta_i : r_i\in R^U\}$ represents the failure (κ_i) and repair (η_i) of the server of unreliable resource $r_i\in R^U$. Service completions, failures and repairs are assumed to be beyond the controller's influence.

Let $\xi:Q\rightarrow 2^{\Sigma}$ be a function that, for a given state, returns the set of enabled events. This function is defined for a state, $q\in Q$, as follows:

1. For $P_{j1}\in\Omega_i$, if $C_i - \sum_{p_{jk}\in\Omega_i}(y_{jk}+x_{jk})>0$, then $\alpha_{j1}\in\xi(q)$.
   Events that release new parts into the system are enabled when space is available on the first required workstation in the route.
2. For $P_{jk}\in\Omega_i$, if $y_{jk}>0$ and $sv_i=1$, then $\beta_{jk}\in\xi(q)$.
   If a part is at service, then the corresponding service completion event is enabled.
3. For $r_i\in R^U$, $P_{jk}\in\Omega_i$ and $\beta_{jk}\in\xi(q) \Rightarrow \kappa_i\in\xi(q)$.
   If the server is busy with a part, then the corresponding failure event is enabled.
4. For $r_i\in R^U$, if $sv_i = 0$, then $\eta_i\in\xi(q)$ and $\beta_{jk}\notin\xi(q)\ \forall P_{jk}\in\Omega_i$.
   If the server is failed, the corresponding repair event is enabled and the corresponding service completion events are disabled.
5. For $P_{jk}\in\Omega_i$, $1<k\leq|P_j|$, if $x_{j,k-1}>0$ and $C_i - \sum_{p_{jk}\in\Omega_i}(y_{jk}+x_{jk})>0$, then $\alpha_{jk}\in\xi(q)$.
   When a part finishes its current operation and buffer space becomes available at the next required workstation in its route, the event corresponding to advancing the part is enabled.
6. For $P_{j,|Pj|}\in\Omega_i$, if $x_{j,|Pj|}>0$, then $\alpha_{j,|Pj|+1}\in\xi(q)$.
   If a part has finished all of its operations, the event corresponding to unloading it from the system is enabled.

The state transition function is now defined as follows. The transition function, δ, is a partial function from the cross product $Q\times\Sigma$ to the set Q of system states. Specifically, let $\delta:Q\times\Sigma\rightarrow Q$ such that

$\delta(q,\alpha_{jk})=q-e_{x_{j,k-1}}+e_{y_{jk}}$, advancing a part $p_{j,k-1}$;

$\delta(q,\beta_{jk})=q-e_{y_{jk}}+e_{x_{jk}}$, service completion of a part p_{jk};

$\delta(q,\kappa_i)=q-e_{sv_i}$, failure of server i;

$\delta(q,\eta_i)=q+e_{sv_i}$, repair of server i;

where $e_{x_{j,k-1}}$, $e_{y_{jk}}$, $e_{x_{jk}}$, and e_{sv_i} are the standard unit vectors with components corresponding to $x_{j,k-1}$, y_{jk}, x_{jk} and sv_i being 1, respectively. Note that, $e_{y_{j,|Pj|+1}} = e_{x_{j0}} = \mathbf{0}$, the zero vector with the same dimension, and that p_{j0} represents a raw part of P_j waiting to be released into the system.

We assume that $|R^U|\geq 1$. In this case, any subset of the unreliable resources can be simultaneously in a failed state. Thus, if one of the $\binom{|R^U|}{i}$ subsets of size i, $i=1\ldots|R^U|$, is down, we want the remaining resources to continue producing parts not requiring any of the failed resources without human intervention to remove or rearrange the parts requiring failed resources. Further, when one of the failed resources is repaired, we want production of parts requiring that resource to resume. A robust controller must possess certain properties in order to accomplish the above-mentioned characteristics.

2.2 Motivating examples for properties of robust supervisory control

This subsection motivates a set of desired properties for a robust controller based upon an example production system. Figure 1 presents an example manufacturing system with two unreliable resources. The stages, routes, and resource capacities are given, as is the complete discrete event model. This model enumerates the resources, capacities, events, and so forth. For now, we will constrain our discussion to the system states presented in Figures 2-4. We recall that, by definition, a resource allocation state is safe if, starting from that state, there exists a sequence of resource allocations/deallocations that completes all parts and takes the system to the empty and idle state, the state in which no resources are allocated and no servers are busy. Our underlying assumption is that if a resource allocation state is safe, then, under correct supervision and starting from that state, it is possible to produce all part types indefinitely.

We have several control objectives for the system of Figure 1. First, we desire that the controller guarantee deadlock-free operation, i.e., that it keeps the system producing all part types. Second, in the event that r_2 fails, we want to continue producing part types not requiring r_2, $\{P_3,P_4\}$, without having to intervene by clearing the system of parts requiring r_2. Similarly, in the event that r_9 fails, we want to continue producing part types not requiring r_9, $\{P_1,P_2,P_4\}$, again without having to intervene by clearing the system of parts requiring r_9. Further, if both r_2 and r_9 are in the failed state, we want to continue producing part types not requiring r_2 or r_9, $\{P_4\}$, again without explicit intervention.

Consider for example the state given in Figure 2. This state is safe; however, if r_2 fails while processing part p_{27} in this state, the production of both P_3 and P_4 will be blocked by two p_{23}'s at r_4. Note that if we advance a p_{23} from r_4 to r_6, then production of P_4 can proceed. However, production of P_3 will now be blocked. Thus, this state does not satisfy our condition that after the failure of r_2, we should be able to continue producing both P_3 and P_4. As another example, consider the state of Figure 3. Again, we see that this state is safe. However, if r_9 fails while processing part p_{35} in this state, production of part types P_1 and P_2 will be blocked by p_{34} at r_6, although the production of P_4 is unaffected.

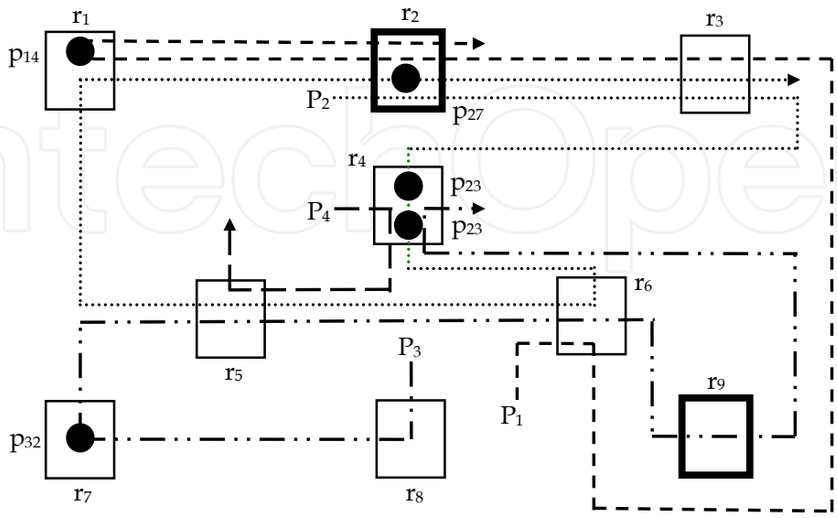


Fig. 2. An undesirable system state since unreliable resource r_2 may fail while processing part p_{27}

Thus, these examples illustrate that parts requiring a failed resource can prevent the system from producing parts not requiring the failed resource through propagation of blocking. Our objective is to develop supervisory controllers that avoid this by guaranteeing that if an unreliable resource fails, it is possible to redistribute the parts requiring that resource so that part types not requiring that resource can continue to produce.

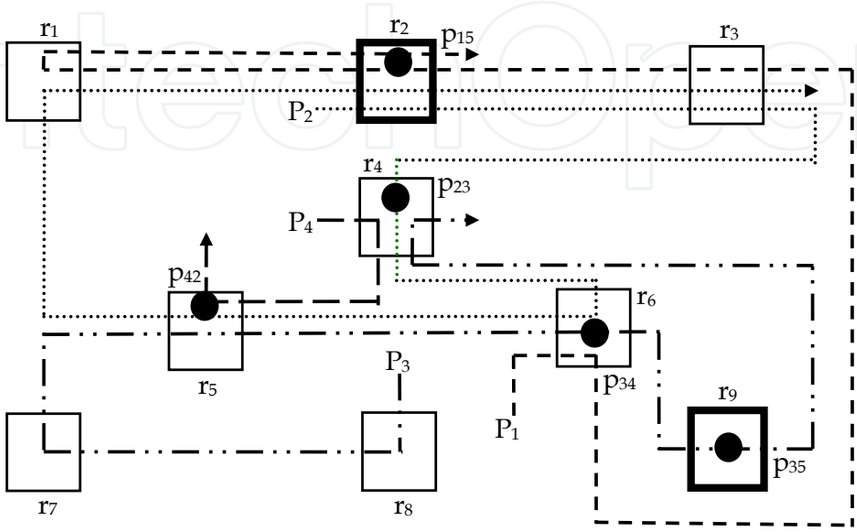


Fig. 3. An undesirable system state since unreliable resource r_9 may fail while processing part p_{35}

For the third objective, consider the state of Figure 4. Again, we see that the state is safe. If r_2 fails, production of P_3 is blocked by p_{11} at r_6. Further, production of P_4 is blocked by p_{33} at r_5. We note that by advancing p_{11} from r_6 to its next required resource, r_3, the blockages of P_3 and P_4 can now be resolved and thus the system can continue producing both P_3 and P_4, as desired. However, when r_2 is repaired, the system is no longer safe since resources r_2 and r_3 are now involved in deadlock. This illustrates that our controller must guarantee that any redistribution of parts requiring the failed resource does not result in system deadlock when the resource is repaired.

The above discussion lays a foundation for a robust supervisory controller. In summary, a supervisory controller is said to be robust to resource failures of R^U if the supervisory controller satisfies Property 2.2.

Property 2.2:

2.2.1: The supervisory controller ensures continuing production of part types not requiring failed resources, given that additional failures/repairs do not occur.

2.2.2: The supervisory controller allows only those states that serve as feasible initial states if an additional resource failure occurs.

2.2.3: The supervisory controller allows only those states that serve as feasible initial states if a failed resource is repaired and becomes operational.

We say that a state is a *feasible initial state* if, starting from that state, it is possible to produce all part types not requiring failed resources. The formal development and definition of this property using language theory is presented in Chew and Lawley (2006).

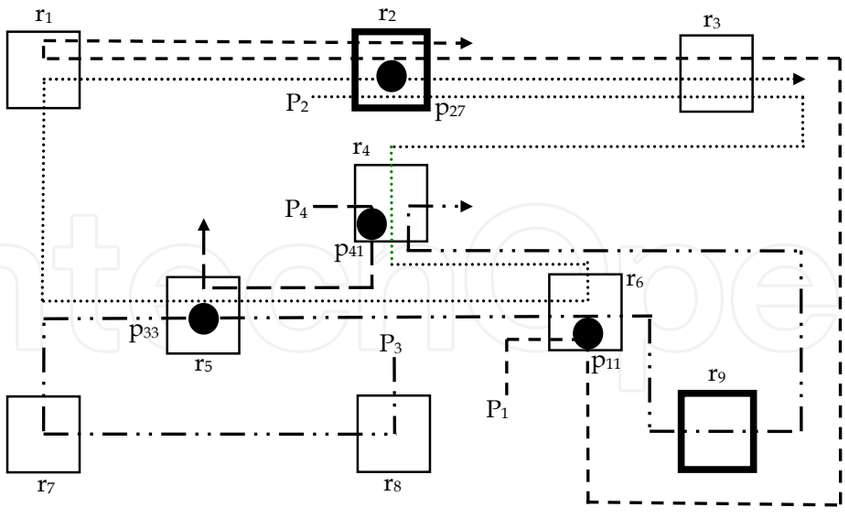


Fig. 4. An undesirable system state since r_2 may fail while processing part p_{27}

3. Robust control for systems with multiple unreliable resources

This section endeavours to delve into robust control for single unit resource allocation systems with unreliable resources.

3.1 Robust control using a neighbourhood policy

This subsection develops controllers that satisfy Property 2.2 above, while maintaining polynomial complexity. Each controller is a conjunction of a modified deadlock avoidance policy and a set of neighbourhood constraints. The deadlock avoidance policy guarantees deadlock-free operation, while the neighbourhood constraints control the distribution of parts that require unreliable resources. Subsection 3.1.1 develops the neighbourhood constraints, NHC. Subsection 3.1.2 constructs a supervisor based on a modified Banker's Algorithm and NHC, while subsection 3.1.3 develops a supervisor based on single-step look-head (SSL) and NHC. The complete proofs can be found in Chew and Lawley (2006).

3.1.1 A neighbourhood policy

In this subsection, we discuss neighbourhood constraints based on the notion of *failure dependency*. Informally, a resource is failure-dependent if every part that enters its buffer space requires some future processing on an unreliable workstation. Thus, all unreliable resources are failure-dependent. Some reliable resources may also be failure-dependent if they only process parts that require future processing on an unreliable resource. This is defined more precisely later. For each failure-dependent resource, we generate a neighbourhood. The neighbourhood of a failure-dependent resource is a virtual space of finite capacity that is used to control the distribution of parts requiring that failure-dependent resource. Again, this is formalized in the following, where we extend the neighbourhood concepts presented by Lawley & Sulistyono (2002) for systems with multiple unreliable resources. We first discuss and illustrate neighbourhood concepts, and then illustrate how neighbourhood constraints are constructed for failure-dependent resources.

Recall that R^U is the set of unreliable resources in the system S, and that $\Omega_i=\{P_{jk}:\rho(P_{jk})=r_i\in R\}$ is the set of part type stages supported by resource r_i. If $r_i\in R^U$, then $r_v\in R$ is said to be *failure-dependent* on r_i if $\forall P_{jk}\in\Omega_v$, $\exists P_{j,k+c}\in\Omega_i$ with $c\geq 0$. In other words, r_v is failure-dependent on r_i if every part that enters the buffer of r_v requires future processing on unreliable resource r_i (note that r_i is failure-dependent on itself). For $r_i\in R^U$, let $R_i^{FD} = \{r_v : r_v\in R$ and $\forall P_{jk}\in\Omega_v$, $\exists P_{j,k+c}\in\Omega_i$ with $c\geq 0\}$ be the set of failure-dependent resources on r_i, and let $R^{FD} = \bigcup_{r_i\in R^u} R_i^{FD}$ and $R^{NFD} = R \setminus R^{FD}$.

For each failure-dependent resource of R_i^{FD}, we construct a neighbourhood. The neighbourhood of $r_v \in R_i^{FD}$, NH_v^i, is defined as the set of part type stages that require r_v now or later in their processing and have no intervening failure-dependent resources of R_i^{FD}. Formally, $NH_v^i = \Omega_v \cup \{P_{jk}$: $\exists c>0$ with $P_{j,k+c}\in\Omega_v$ and $\forall d\in[0,c)$, $\rho(P_{j,k+d}) \notin R_i^{FD}\}$. Thus, if $\rho(P_{j,k+c}) = r_v \in R_i^{FD}$, $\rho(P_{j,k-1}) = r_w \in R_i^{FD}$, with $r_v\neq r_w$, and $\{\rho(P_{jk}), \rho(P_{j,k+1})\ldots\rho(P_{j,k+c-1})\} \cap R_i^{FD} = \varnothing$, then $\{P_{jk},P_{j,k+1} \ldots P_{j,k+c-1},P_{j,k+c}\} \subseteq NH_v^i$, and $P_{j,k-1} \notin NH_v^i$. Let $NH^i = \{NH_v^i : r_v\in R_i^{FD}\}$ be the neighbourhood set for $r_i\in R^U$, and let $NH=\{NH^i : r_i\in R^U\}$.

For example, the system of Figure 1 has two unreliable resources, $R^U=\{r_2,r_9\}$. Note that anytime r_1 appears in a route, r_2 appears later in the route, and thus, $R_2^{FD} = \{r_1,r_2\}$. Also, anytime r_7 or r_8 appear in a route, r_9 appears later in the route, so, $R_9^{FD} = \{r_7,r_8,r_9\}$. Thus, $NH^2 = \{NH_1^2, NH_2^2\}$ and $NH^9 =\{NH_7^9, NH_8^9, NH_9^9\}$, where the neighbourhoods are as follows: $NH_1^2 =\{P_{14},P_{22},P_{23},P_{24},P_{25},P_{26}\}$, $NH_2^2 = \{P_{11},P_{12},P_{13},P_{15},P_{21},P_{27}\}$, $NH_7^9 = \{P_{32}\}$, $NH_8^9 = \{P_{31}\}$, $NH_9^9 = \{P_{33},P_{34},P_{35}\}$.

To understand this construction, consider NH_1^2 and NH_2^2. Note that $\Omega_1= \{P_{14},P_{26}\}$ and $\Omega_2=\{P_{13},P_{15},P_{21},P_{27}\}$. Since $\Omega_v \subseteq NH_v^i$, $\{P_{14},P_{26}\} \subseteq NH_1^2$, and $\{P_{13},P_{15},P_{21},P_{27}\} \subseteq NH_2^2$. Now consider $T_1 = \{\rho(P_{11}),\rho(P_{12}),\rho(P_{13}),\rho(P_{14}),\rho(P_{15})\} = \{r_6,r_3,r_2,r_1,r_2\}$. Since $\{r_6,r_3\} \cap R_2^{FD} = \varnothing$, $\{P_{11},P_{12}\} \subseteq NH_2^2$. Similarly, $T_2 = \{\rho(P_{21}),\rho(P_{22}),\rho(P_{23}),\rho(P_{24}),\rho(P_{25}),\rho(P_{26}),\rho(P_{27}),\rho(P_{28})\} = \{r_2,r_3,r_4,r_6,r_5,r_1,r_2,r_3\}$. Since $\{r_3,r_4,r_6,r_5\} \cap R_2^{FD} = \varnothing$, $\{P_{22},P_{23},P_{24},P_{25}\} \subseteq NH_1^2$. Thus, we get $NH_1^2 = \{P_{14},P_{22},P_{23},P_{24},P_{25},P_{26}\}$ and $NH_2^2 = \{P_{11},P_{12},P_{13},P_{15},P_{21}, P_{27}\}$.

Although all parts supported by r_6 later need an unreliable resource, r_6 is shared by r_2 and r_9, and thus it is not failure-dependent on either. This implies that failure-dependent sets are disjoint, i.e., $R_2^{FD} \cap R_9^{FD} = \varnothing$. Furthermore, we observe that no part stage is in more than one neighbourhood, i.e., $NH_1^2 \cap NH_2^2 \cap NH_7^9 \cap NH_8^9 \cap NH_9^9 = \varnothing$. These and other important neighbourhood properties are established in Chew and Lawley (2006).

We restrict the number of parts allowed in a neighbourhood. Our intention is to guarantee that every part in the neighbourhood of a failure-dependent resource has capacity reserved at that resource. That is, we want to be able to advance every part requiring an unreliable resource into its associated failure-dependent resource in the event of a resource failure so that it will not block production of parts not requiring the failed resource. In the example, for a permissible state, we want, for example, every part in $NH_9^9 = \{P_{33},P_{34},P_{35}\}$ to have a reserved unit of buffer at r_9. As a consequence, we will reject a state if this constraint is violated. For instance, a state is not admissible if, at this state, the sum of parts in $NH_9^9 > 1$; recall that r_9 has a single unit of capacity. To see this, at this inadmissible state, if r_9 fails, at least one part of NH_9^9 must reside at r_5 or r_6. Although P_1, P_2, and P_4 do not require failed r_9 in their processing, this distribution of parts may in turn block production of some of these part types. Our objective is to develop supervisory controllers capable of rejecting these undesirable states.

We now construct neighbourhood constraints to enforce the above intention. The constraint for a neighbourhood, say NH_v^i, is an inequality of the form $Z_v^i \le C_v$ where $Z_v^i = \sum_{P_{jk} \in NH_v^i} (x_{jk} + y_{jk})$. Recall that x_{jk} is the number of finished instances, and y_{jk} is the number of unfinished instances, of P_{jk} located in the buffer of $\rho(P_{jk})$; and that the right hand side C_v is the capacity of r_v. NH_v^i is said to be *capacitated* if $Z_v^i = C_v$ and *over-capacitated* if $Z_v^i > C_v$. Define the set of all possible neighbourhood constraints with respect to $r_i \in R^U$ as:

$$NHC_1^i = \{ Z_v^i \le C_v : NH_v^i \in NH^i \}.$$

In the example, we have

$$NHC_1^2 = \{ Z_1^2 = \sum_{P_{jk} \in NH_1^2} (x_{jk} + y_{jk}) \le C_1,\ Z_2^2 = \sum_{P_{jk} \in NH_2^2} (x_{jk} + y_{jk}) \le C_2 \};$$

$$NHC_1^9 = \{ Z_7^9 = \sum_{P_{jk} \in NH_7^9} (x_{jk} + y_{jk}) \le C_7,\ Z_8^9 = \sum_{P_{jk} \in NH_8^9} (x_{jk} + y_{jk}) \le C_8,$$

$$Z_9^9 = \sum_{P_{jk} \in NH_9^9} (x_{jk} + y_{jk}) \le C_9 \}.$$

Constraints of NHC_1^i assure that no neighbourhood of NH^i becomes over-capacitated.
As Lawley & Sulistyono (2002) discuss, NHC_1^i may induce deadlock among failure-dependent resources of R_i^{FD}, since if all neighbourhoods are capacitated, parts cannot move from one neighbourhood to another without over-capacitating a neighbourhood. In the example, a state may satisfy both $1 = Z_1^2 \le C_1 = 1$ and $1 = Z_2^2 \le C_2 = 1$. But, a part moves from one of these associated neighbourhoods to another must over-capacitate the other neighbourhood. To resolve this dilemma, we develop an additional set of constraints, NHC_2^i.
It is first necessary to compute the set of strongly connected neighbourhoods for NHC_2^i. To do this, for each $r_i \in R^U$, we construct a directed graph (NH^i, A^i) where $A^i = \{(NH_g^i, NH_h^i) : \exists P_{jk} \in NH_g^i$ with $P_{j,k+1} \in NH_h^i \}$. Thus, in operation, there will be part flow from NH_g^i to NH_h^i. We then compute the set of strong components of (NH^i, A^i) using standard polynomial graph algorithms (Cormen et at., 2002). For example, we see that NH_1^2 and NH_2^2 are strongly connected, since $\{P_{14}, P_{26}\} \subseteq NH_1^2$ and $\{P_{13}, P_{27}\} \subseteq NH_2^2$. Therefore, in operation, there is flow from NH_1^2 to NH_2^2 and from NH_2^2 to NH_1^2. Let SC^i be the set of strongly connected components of (NH^i, A^i). Then, $SC^2 = \{ SC_1^2 = \{ NH_1^2, NH_2^2 \} \}$, and $SC^9 = \{ SC_1^9 = \{ NH_7^9 \}, SC_2^9 = \{ NH_8^9 \}, SC_3^9 = \{ NH_9^9 \} \}$. Then, NHC_2^i is stated as follows:

$$NHC_2^i = \{ Z_g^i + Z_h^i < C_g + C_h : \{ NH_g^i, NH_h^i \} \subseteq SC_m^i \in SC^i,\ m = 1 \ldots |SC| \}.$$

Hence, for every strongly connected component of (NH^i, A^i), NHC_2^i guarantees that at most one neighbourhood can be capacitated at a time. In the example, we have the following:

$$NHC_2^2 = \{ Z_1^1 + Z_2^2 < C_1 + C_2 \},\ NHC_2^9 = \varnothing.$$

NHC_2^2 guarantees that NH_1^2 and NH_2^2 are not simultaneously capacitated.
To summarize, $NHC^i = NHC_1^i \cup NHC_2^i$ guarantees that no neighbourhood is over capacitated, and that neighbourhoods with mutual flow dependencies are not simultaneously capacitated. The complete set of neighbourhood constraints is defined as:

$$NHC = \{ NHC^i : r_i \in R^U \}.$$

Note that in the worst case, we generate one constraint for each pair of resources and thus the size of NHC is of $O(|R|^2)$.
Chew and Lawley (2006) establish several important properties of NHC. These properties are required to establish robustness of the two supervisors that we develop later. We next modify two deadlock avoidance policies that we use in conjunction with NHC to develop robust supervisors.

3.1.2 Banker's algorithm

In this subsection, we configure Banker's Algorithm (BA) (Habermann, 1969) to work with NHC. BA is perhaps the most widely known deadlock avoidance policy (DAP), and its underlying concepts have influenced the thinking of numerous researchers. BA is a suboptimal DAP in the sense that it achieves computational tractability by sacrificing some safe states. BA avoids deadlock by allowing an allocation only if the requesting processes can be ordered so that the terminal resource needs for the i^{th} process, P_i, in the ordering can be met by pooling available resources and those released by completed processes P_1, P_2 … P_{i-1}. The ordering is essentially a sequence in which all processes in the system can complete successfully. BA is of O(mnlog n) where m is the number of resource types and n is the number of requests. Other manufacturing related work also uses BA (Ezpeleta et al., 2002; Lawley et al., 1998; Reveliotis, 2000).
Our modifications are straightforward and are a generalization of those undertaken by Lawley & Sulistyono (2002). Our objective is to search for an ordering of parts that advances failure-dependent parts (those requiring unreliable resources) into the resource of their current neighbourhood, and non-failure-dependent parts (those not requiring unreliable resources) out of the system. Again, the ordering is such that the resources required by the first part are all available, those required by the second part are all available after the first part has finished and released the resources held by the part, and so forth. If the system can be cleared in this way (all failure-dependent parts are advanced into failure-dependent resources and all non-failure-dependent parts are advanced out of the system), then we can guarantee that if any unreliable resource fails, the system can continue producing parts that do not require this failed resource.
In the following, let $\Lambda = \Lambda^{NFD} \cup \Lambda^{FD}$ be the set of part type stages instantiated in q *whose parts hold non-failure-dependent resources*, where Λ^{NFD} is the set of non-failure-dependent part type stages (those that do not require failure-dependent resources in the residual route) and Λ^{FD} is the set of failure-dependent part type stages (those that do require failure-dependent resources in the residual route). We now present our modified version of BA as Algorithm A1 as follows.

Algorithm A1:

Query: Is state q admissible?
Input: state q;
Output: ACCEPT / REJECT
Step 1: Initialization
For P(u)
  For r_v R
    ALLOCATION[u][v]=0
    NEED[u][v]=0
    AVAILABLE[v]=C_V
  End For
End For
For P(u)=P_{jk} and r_v = (P_{jk})
  ALLOCATION[u][v]=$x_{jk}+y_{jk}$
End For
For P(u)=P_{jk} NFD
  For r_v $RT_{jk}\{$ (P_{jk})}
    NEED[u][v]=1
  End For
End For
For P(u)=P_{jk} FD
  For r_v RT_{jk}
    NEED[u][v]=1
  End For
  For r_v RT_{jk} R^{FD}
    NEED[u][v]=0
  End For
  For r_s= (P_{jc}) and r_t= ($P_{j,c+1}$), c=k….|P_j| 1

  NEED[u][t]=NEED[u][s]*NEED[u][t]
  End For
  For r_v RT_{jk}
    If r_v= (P_{jk})
      NEED[u][v]=0
    End If
  End For
End For
For r_v R^{NFD}
  AVAILABLE[v]=C_V $_u$ALLOCATION[u][v]
End For
 =

Step 2: Test and Evaluation
While
  Find P(u) such that NEED[u][v] AVAILABLE[v] for all r_v R^{NFD}
  If no such P(u) exists
    Return REJECT
  Else
    = \{P(u)}

    AVAILABLE[v]=AVAILABLE[v]+ALLOCATION[u][v] for r_v= (P(u))
  End If
End While
Return ACCEPT

Step 1 of the algorithm configures the data structures required. For every part type stage represented in the system, these capture the current resource holding and the future processing need. These structures also capture the resource availability of resources in the state being tested. Three additional comments regarding the algorithm are in order. First, the need for every failure-dependent resource is explicitly set to zero, so this version looks only at the availability of non-failure-dependent resources. Second, for non-failure-dependent part type stages (those not requiring unreliable resources), the need for every resource in the residual route (except the one held) is set to one. Finally, for failure-dependent part type stages (those requiring unreliable resources), the need for every resource in the residual route (except the one held) up to the one immediately preceding the first encountered failure-dependent resource is set to one, all others are set to zero. Note that these are the resources such a part will need to advance into the failure-dependent resource of its current neighbourhood. Step 2 then executes the usual Banker's logic.

Algorithm A1 is not correct by itself, since it does not handle allocation of failure-dependent resources (for detailed examples the reader is referred to the work by Lawley & Sulistyono

(2002)). However, A1 and NHC together form a robust controller, that is, if we allow the system to visit only those states acceptable to both A1 and NHC, then the system operation will satisfy the requirements of Property 2.2. The detailed proofs for this are given in Chew and Lawley (2006). The supervisor is defined as follows:

Definition 3.1.1: Supervisor Δ_1 = A1 $\wedge$ NHC.

Supervisor Δ_1 permits a system state that satisfies both A1 and NHC in runtime. Consider Figure 5, which illustrates a state, say q, in which r_4 holds p_{23} and p_{14}; and r_5 holds p_{33}. It is clear, at q, that there exists an admissible sequence by A1; that is, p_{33} can advance into failure-dependent resource r_9; p_{23} can advance into failure-dependent resource r_1; and finally, p_{41} can be advanced out of the system. In addition, q satisfies NHC since

$$NHC_1^2 = \{ Z_1^2 = 1 \le 1, \text{(there is one } P_{23}) \quad Z_2^2 = 0 \le 1\};$$

$$NHC_1^9 = \{ Z_7^9 = 0 \le 1, \quad Z_8^9 = 0 \le 1, \quad Z_9^9 = 1 \le 1 \text{ (there is one } P_{33})\};$$

$$NHC_2^2 = \{1 + 0 < 1 + 1 = 2\}; \; NHC_2^9 = \varnothing.$$

Therefore, q is an admissible state by Δ_1. Supervisor Δ_1 will prohibit, at q, advancing p_{23} into r_6 (where it becomes p_{24}) because p_{24} and p_{33} will block, causing the resulting state to violate A1 although not NHC. Loading a p_{11} into r_6 at q is also precluded by Δ_1 since the resulting state violates NHC_2^2 , although not A1. However, advancing p_{33} one step into r_6 or loading a p_{31} into r_8 will result in an admissible state.

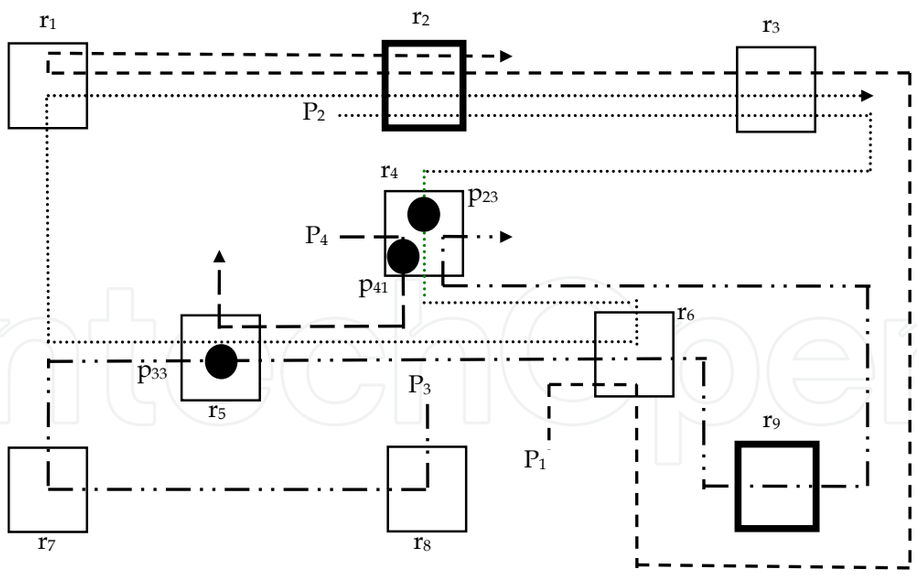


Fig. 5. An admissible system state by supervisor Δ_1

Supervisor Δ_1 is of polynomial complexity since both A1 and NHC require polynomial time for runtime implementation. Chew and Lawley (2006) formally establish that Δ_1 yields a robust supervisor for systems where every part type requires in its route at most one unreliable resource.

3.1.3 A Single step look ahead policy

It is well known that certain system structures, such as a central buffer, input/output bins, and non-unit buffer capacities, eliminate the possibility of deadlock-free unsafe states (Lawley & Reveliotis, 2001). In these systems, every state is either deadlock or safe, and therefore, a single-step look-ahead policy (SSL) is a correct and optimal deadlock avoidance policy. Further, it is of polynomial complexity, and thus ideal for runtime applications in real systems. In the following, we will modify the SSL presented by Lawley (1999) so that it works with systems with multiple unreliable resources.

A resource allocation graph (RAG) is a digraph that encodes the resource requests and allocations of parts (Lawley, 1999). For our purposes, let RAG=$(R\backslash R^{FD},E)$ where $R\backslash R^{FD}$ is the set of system non-failure-dependent resource types and $E=\{(r_u,r_v): r_u,r_v\in R\backslash R^{FD}$ and r_u is holding a part p_{jk} with $\rho(p_{j,k+1})=r_v\}$. A subdigraph of RAG, say (R,E), is *induced* when $R \subseteq R\backslash R^{FD}$ and $E=\{(r_u,r_v):(r_u,r_v)\in E$ and $r_u,r_v\in R\}$. A subdigraph, (R,E), forms a *knot* in RAG if $\forall r_u\in R$, $\Gamma(r_u)= R$, where $\Gamma(r_u)$ is the set of all nodes reachable from r_u in RAG. In other words, a set of nodes, R, forms a knot in RAG when, for every node in R, the set of nodes reachable along arcs in RAG is exactly R. Further, we define a *capacitated knot* to be a knot in which every resource in the knot is filled to capacity with parts requesting other resources in the knot. It is commonly known that a capacitated knot in RAG is a necessary and sufficient condition for deadlock in these types of sequential resource allocation systems. We now provide an algorithm, Algorithm A2, below to detect a capacitated knot in RAG = $(R\backslash R^{FD},E)$. This algorithm has the same polynomial complexity as that given by Lawley (1999).

Algorithm A2:

Input: RAG=$(R\backslash R^{FD},E)$

Output: DEADLOCK, NO DEADLOCK

Step 1: Compute the set of strongly connected components of RAG: $C=\{C_1\ldots C_q\}$

Step 2: Construct digraph (C,E_c) such that $C=\{C_1\ldots C_q\}$ and $E_c=\{(C_i,C_j):(r_u,r_v)\in E$ with $r_u\in C_i$ and $r_v\in C_j$ for $i\neq j\}$

Step 3: For every strongly connected component $C_i\in C$ such that $(C_i,C_j)\notin E_c$ $\forall j=1\ldots q$

```
If Ci is a capacitated knot
   Return DEADLOCK
End If
End For
```

Step 4: Return NO DEADLOCK

We note that, for our present work, this version of deadlock detection algorithm operates only on non-failure-dependent resources and parts held by these resources. In A2, Step 1 computes the set of strongly connected components in RAG. As mentioned earlier, this is a standard digraph operation. Step 2 constructs a digraph that defines the reachability relationship between these components. Step 3 looks for a component with no outgoing arc. If such a component is filled to capacity with parts requesting other resources in the component, then it is a capacitated knot, and deadlock exists. If no such capacitated knot exists then the RAG is deadlock-free.

Note that A2 is not correct by itself since it considers only the non-failure-dependent resources. Failure-dependent resources can easily deadlock themselves. However, when A2 is taken in conjunction with NHC, it guarantees Property 2.2 and thus assures that the system will continue to operate even when multiple unreliable resources are down.

Definition 3.1.2: Supervisor Δ_2 = A2 $\wedge$ NHC.
Supervisor Δ_2 accepts a system state that contains no deadlock and satisfies NHC. For example, in Figure 1, suppose that every non-failure-dependent resource has non-unit capacity; that is, $C_i>1$, $\forall r_i \in R \setminus R^{FD} = \{r_3,r_4,r_5,r_6\}$. Then, A2 permits any state in which no subset of parts residing on $\{r_3,r_4,r_5,r_6\}$ is deadlocked on $\{r_3,r_4,r_5,r_6\}$. If the state also satisfies NHC, then Property 2.2 is guaranteed.
Note that Δ_2 = A2 $\wedge$ NHC is suited for real-time implementation since both A2 and NHC are of polynomial complexity. Chew and Lawley (2006) formally establishes that Δ_2 yields a robust supervisor for systems where every part type requires in its route at most one unreliable resource.

3.2 Robust control using a resource order policy

This subsection configures a deadlock avoidance policy, resource order policy (RO). We will employ this configured resource order policy in conjunction with the neighbourhood constraints of Subsection 3.1 to develop a robust controller. Consider, for configuration purposes, Figure 1. Define RCO = $R \setminus R_i^{FD}$ as the set of non-failure-dependent resources. Since $R_2^{FD} = \{r_1,r_2\}$ and $R_9^{FD} = \{r_7,r_8,r_9\}$, thus RCO = $\{r_3,r_4,r_5,r_6\}$. Let ω : RCO $\rightarrow \aleph$ (the set of natural numbers) be a one to one mapping of non-failure-dependent resources (ω orders the non-failure-dependent resources so that RO can be applied); $P^{FD}=\{P_j: \rho(P_{jk}) \in R^U \text{ for some } k\}$ (P^{FD} is the set of part types requiring unreliable resources; thus, in Figure 1, $P^{FD} = \{P_1,P_2,P_3\}$); and $P^{NFD}=P \setminus P^{FD}$ (P^{NFD} is the set of part types not requiring any unreliable resources; hence, in Figure 1, $P^{NFD} = \{P_4\}$). For each $P_j \in P^{FD}$, determine all maximal subsequences in the route of P_j that do not contain failure-dependent resources. For instance, in Figure 1, $P_3 \in P^{FD}$ where $P_3=\langle P_{31}, P_{32}, P_{33}, P_{34}, P_{35}, P_{36}\rangle$ with route $\langle r_8,r_7,r_5,r_6,r_9,r_4\rangle$, the maximal subsequences in $\langle r_8,r_7,r_5,r_6,r_9,r_4\rangle$ that do not contain failure-dependent resources are $\langle r_5,r_6\rangle$ and $\langle r_4\rangle$.
To express this formally, for each $P_j \in P^{FD}$, break the route of P_j into subroutes as follows: for $P_j=\langle P_{j1} \ldots P_{j,k_1-1},P_{jk_1},P_{j,k_1+1} \ldots P_{j,k_2-1},P_{jk_2},P_{j,k_2+1} \ldots P_{j,k_{h_j}-1},P_{jk_{h_j}},P_{j,k_{h_j}+1} \ldots\rangle$, $\{P_{jk_1},P_{jk_2} \ldots P_{jk_{h_j}}\}$ being precisely the set of part type stages of P_j that is processed on failure-dependent resources (that is, $\{\rho(P_{jk}) : k=k_1,k_2 \ldots k_{h_j}\} \subseteq R^{FD}$ and $\{\rho(P_{jk}) : k \neq k_1,k_2 \ldots k_{h_j}\} \cap R^{FD} = \varnothing$), let $P_j^1 = \langle P_{j1} \ldots P_{j,k_1-1}\rangle$, $P_j^2 = \langle P_{j,k_1+1} \ldots P_{j,k_2-1}\rangle$, $P_j^3 = \langle P_{j,k_2+1} \ldots P_{j,k_3-1}\rangle, \ldots$, $P_j^{h_j} = \langle P_{j,k_{(h_j-1)}+1} \ldots P_{j,(k_{h_j}-1)}\rangle$ and $P_j^{h_j+1} = \langle P_{j,(k_{h_j}+1)} \ldots P_{j|P_j|}\rangle$. For each $P_j \in P^{NFD}$, rename P_j P_j^0. Finally, let $P' = \{P_j^0 : P_j \in P^{NFD}\} \cup \{P_j^k : k = 1 \ldots h_j \text{ and } P_j \in P^{FD}\}$. Note that in P′, a part type $P_j \in P^{FD}$ is replaced by a set of part types $\{P_j^1, P_j^2 \ldots P_j^{h_j}\}$ each having a route that is a maximal segment of the route of P_j not containing a failure-dependent resource.
In Figure 1, for example, P_3 is replaced by $P_3^1 = \langle P_{33},P_{34}\rangle$ with route $\langle r_5,r_6\rangle$ and $P_3^2 = \langle P_{36}\rangle$ with route $\langle r_4\rangle$, and P_4 is renamed P_4^0. Thus, the revised set of part types is $P'=\{P_4^0\} \cup \{P_1^1, P_2^1, P_2^2, P_3^1, P_3^2\}$. Note that none of the routes of part types in P′ contains any failure-dependent resources.
We now use P′ and RCO to construct a set of RO constraints as follows. For each $P_j^i = \langle P_{j,k_{(i-1)}+1} \ldots P_{j,(k_i-1)}\rangle \in P'$ and for each $P_{jk} \in P_j^i$, consider the inclusive remaining route, $\langle \rho(P_{jk}) \ldots \rho(P_{j,(k_i-1)})\rangle$, and its mapping, $\langle \omega(\rho(P_{jk})) \ldots \omega(\rho(P_{j,(k_i-1)}))\rangle$. (Recall that to implement RO, the resources must be ordered. ω represents the ordering function.) If the mapping of the inclusive remaining route is strictly increasing (decreasing), then P_{jk} is classified as 'right' ('left'); if the mapping of the inclusive remaining route switches direction at some point,

then P_{jk} is classified as 'undirected.' If P_{jk} is terminal, it is ignored. For $r_m \in RCO$, let Π_m^{RU} represent the set of right and undirected part type stages associated with r_m; and Π_m^{LU}, the set of left and undirected part type stages associated with r_m. In the example, consider that $\omega(r_1)=1$, $\omega(r_2)=2$, $\omega(r_3)=3$ $\omega(r_4)=4$, $\omega(r_5)=5$, $\omega(r_6)=6$, $\omega(r_7)=7$, $\omega(r_8)=8$ and $\omega(r_9)=9$. We now have that the inclusive remaining route of P_{33}, $\langle r_5,r_6\rangle$ supporting $\langle P_{33},P_{34}\rangle \subseteq P_3^1$, is strictly increasing for ω, thus P_{33} is classified as 'right' and hence $\Pi_5^{RU} = \{P_{33}\}$. In the meantime, since $P_{25} \in P_2^1$ is the terminal part type stage for P_2^1, P_{25} is ignored. Clearly, $\Pi_5^{RU} = \varnothing$. On the other hand, the inclusive remaining route of P_{11}, $\langle r_6,r_3\rangle$ supporting $\langle P_{11},P_{12}\rangle \subseteq P_1^1$, is strictly decreasing for ω, hence P_{11} is classified as 'left.' The inclusive remaining route of P_{24} is $\langle r_6,r_5\rangle$ supporting $\langle P_{24},P_{25}\rangle \subseteq P_2^1$, which is strictly decreasing for ω, hence P_{24} is classified as 'left.' Therefore, $\Pi_6^{LU} = \{P_{11},P_{24}\}$. Meanwhile, since $P_{34} \in P_3^1$ is the terminal part type stage for P_3^1, P_{34} is ignored. It is obvious that $\Pi_6^{LU} = \varnothing$. After all the part type stages are classified in this way, a constraint is generated for each pair of non-failure-dependent resources, yielding RO constraints. We now define RO constraints formally as follows.

Definition 3.2.1: RO^{RCO} is the set of constraints:

$$\forall r_m, r_n \in RCO \text{ such that } \omega(r_m) < \omega(r_n), \quad \sum_{P_{jk} \in \Pi_m^{RU}} (x_{jk} + y_{jk}) + \sum_{P_{jk} \in \Pi_n^{LU}} (x_{jk} + y_{jk}) < C_m + C_n$$

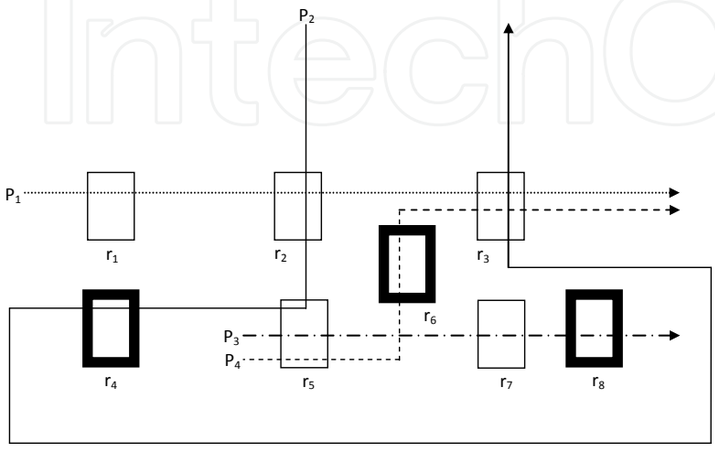
where C_m and C_n are the respective buffer capacities of r_m and r_n.

In the example, for r_5, $r_6 \in RCO$, we have $(x_{33}+y_{33}) + (x_{11}+y_{11}+x_{24}+y_{24}) < 2$, recalling that $C_5=C_6=1$. This constraint assures that for every resource allocation state that the system is allowed to visit, the number of 'right' and 'undirected' parts occupying buffer space at r_5 plus the number of 'left' and 'undirected' parts occupying buffer space at r_6 will be less than the combined capacity of the two resources. Similar constraints are generated for the resource pairs $\{r_3,r_4\}$, $\{r_3,r_5\}$, $\{r_3,r_6\}$, $\{r_4,r_5\}$ and $\{r_4,r_6\}$.

We are now in the position to establish that the conjunction of RO^{RCO} and NHC, call it supervisor Δ_3, satisfies Property 2.2. Supervisor Δ_3 is a control policy such that it disables $\alpha_{jk} \in \xi(q)$ if $\delta(q,\alpha_{jk})$ violates either RO^{RCO} or NHC. Formally, it is stated as follows.

Definition 3.2.2: Supervisor $\Delta_3 = RO^{RCO} \wedge NHC$.

Chew et al. (2011) establish that Δ_3 is a robust controller for systems where every part type requires at most one unreliable resource.



R={r1,r2,r3,r4,r5,r6,r7,r8}
P={P1,P2,P3,P4}
R^U={r4,r6,r8}
P1={P11,P12,P13}
T1=⟨r1,r2,r3⟩
P2={P21,P22, P23,P24}
T2=⟨r2,r5,r4,r3⟩
P3={P31,P32,P33}
T3=⟨r5,r7,r8⟩
P4={P41,P42,P43}
T4={r5,r6,r3}
C=⟨C1,C2,...,C8⟩=⟨2,2,2,2,2,2,2,2⟩

Fig. 6. An example production system with three unreliable resources

3.3 Robust control using shared resource capacity

The robust supervisory control policies presented in sections 3.1-3.2 assume that that parts requiring failed resources can be advanced into FD buffer. We refer this type of control policies as "absorbing" policies. This subsection relaxes this assumption because, in some systems, providing FD buffer space might be too expensive or it might be desirable to load the system more heavily with FD parts. A "distributing" type of control policy is developed and presented in this subsection. This policy distributes parts requiring failed resources throughout the buffer space of shared resources so that these distributed parts do not block the production of part types that are not requiring failed resources.

Now, the development of the "distributing" control policy, namely, RO^4 policy is discussed in details. First, based on the definitions of resource sets in the previous sections, we further define three resource regions: (1) the region of continuous operation, $RCO=R^{PFD}\cup R^{NFD}$, (2) the region of failure dependency, $RFD=R^{FD}$, and (3) the region of distribution, $ROD=RFD\backslash R^U = R^{FD}\backslash R^U = R^R\backslash R^{NFD}$. In the example system in Figure 6, we have $RCO=\{r_1,r_2,r_3\}$; $RFD=\{r_2,r_4,r_5,r_6,r_7,r_8\}$; $ROD=\{r_2,r_5,r_7\}$. RO^4 policy is the conjunction of four modified RO policies applied to different resource regions. We now define the RO constraints as follows.

Definition 3.3.1: RO^{RCO} is the set of constraints:

$$\sum_{P_{jk}\in\Omega_g} z_{jk} + \sum_{P_{uv}\in\Omega_h} z_{uv} < C_g + C_h$$
$$where \quad z_{st} = x_{st} + y_{st}, \quad r_g, r_h \in RCO \text{ and } g \neq h.$$

RO^{RCO} admits states that exhibit at most one capacitated resource in RCO.

Definition 3.3.2: RO^{RFD} is the set of constraints

$$\sum_{P_{jk}\in\Omega_g\cap P_i^{FD}} z_{jk} + \sum_{P_{uv}\in\Omega_h\cap P_i^{FD}} z_{uv} < C_g + C_h \quad for \quad r_i \in R^U$$
$$where \quad z_{st} = x_{st} + y_{st}, \quad r_g, r_h \in RFD, \quad and \quad g \neq h.$$

RO^{RFD} admits states for which at most one resource of RFD is capacitated with P_i^{FD} parts for each $r_i\in R^U$. Note that it does not place any constraint on the total number of RFD resources capacitated.

Definition 3.3.3: RO^{RFD^2} is the set of constraints

$$\sum_{P_{jk}\in\Omega_g\cap P^{FD}} z_{jk} + \sum_{P_{mn}\in\Omega_h\cap P^{FD}} z_{mn} + \sum_{P_{uv}\in\Omega_j\cap P^{FD}} z_{uv} < C_g + C_h + C_j$$
$$where \quad z_{st} = x_{st} + y_{st}, \quad r_g, r_h, r_j \in RFD \text{ and } g \neq h \neq j.$$

RO^{RFD^2} admits states for which at most two resources of RFD are capacitated with FD parts, but does not place any constraint on the total number of RFD resources capacitated.

Definition 3.3.4: RO^{ROD} is the set of constraints

$$\sum_{P_{jk}\in\Omega_g\cap P^{FD}} z_{jk} + \sum_{P_{uv}\in\Omega_h\cap P^{FD}} z_{uv} < C_g + C_h$$
$$where \quad z_{st} = x_{st} + y_{st}, \quad r_g, r_h \in ROD \text{ and } g \neq h.$$

RO^{ROD} admits states for which at most one resource of ROD=RFD\R^U is capacitated with FD parts, although it places no constraint on the number of unreliable resources that are capacitated.

As in the example system in Figure 6, the set of constrains are as follows.

RO^{RCO}	r_1r_2:	$z_{11}+z_{12}+z_{21}<4$	r_2r_3:	$z_{12}+z_{21}+z_{13}+z_{24}+z_{43}<4$
	r_1r_3:	$z_{11}+z_{13}+z_{24}+z_{43}<4$		
RO^{RFD}	r_2r_4:	$z_{21}+z_{23}<4$	r_5r_7:	$z_{31}+z_{32}<4$
	r_2r_5:	$z_{21}+z_{22}<4$	r_5r_8:	$z_{31}+z_{33}<4$
	r_4r_5:	$z_{23}+z_{22}<4$	r_7r_8:	$z_{32}+z_{33}<4$
	r_5r_6:	$z_{41}+z_{42}<4$		
RO^{RFD^2}	$r_2r_4r_5$:	$z_{21}+z_{23}+z_{22}+z_{31}+z_{41}<6$	$r_4r_5r_6$:	$z_{23}+z_{22}+z_{31}+z_{41}+z_{42}<6$
	$r_2r_4r_6$:	$z_{21}+z_{23}+z_{42}<6$	$r_4r_5r_7$:	$z_{23}+z_{22}+z_{31}+z_{41}+z_{32}<6$
	$r_2r_4r_7$:	$z_{21}+z_{23}+z_{32}<6$	$r_4r_5r_8$:	$z_{23}+z_{22}+z_{31}+z_{41}+z_{33}<6$
	$r_2r_4r_8$:	$z_{21}+z_{23}+z_{33}<6$	$r_4r_6r_7$:	$z_{23}+z_{42}+z_{32}<6$
	$r_2r_5r_6$:	$z_{21}+z_{22}+z_{31}+z_{41}+z_{42}<6$	$r_4r_6r_8$:	$z_{23}+z_{42}+z_{33}<6$
	$r_2r_5r_7$:	$z_{21}+z_{22}+z_{31}+z_{41}+z_{32}<6$	$r_4r_7r_8$:	$z_{23}+z_{32}+z_{33}<6$
	$r_2r_5r_8$:	$z_{21}+z_{22}+z_{31}+z_{41}+z_{33}<6$	$r_5r_6r_7$:	$z_{22}+z_{31}+z_{41}+z_{42}+z_{32}<6$
	$r_2r_6r_7$:	$z_{21}+z_{42}+z_{32}<6$	$r_5r_6r_8$:	$z_{22}+z_{31}+z_{41}+z_{42}+z_{33}<6$
	$r_2r_6r_8$:	$z_{21}+z_{42}+z_{33}<6$	$r_5r_7r_8$:	$z_{22}+z_{31}+z_{41}+z_{32}+z_{33}<6$
	$r_2r_7r_8$:	$z_{21}+z_{32}+z_{33}<6$	$r_6r_7r_8$:	$z_{42}+z_{32}+z_{33}<6$
RO^{ROD}	r_2r_5:	$z_{21}+z_{22}+z_{31}+z_{41}<4$	r_5r_7:	$z_{22}+z_{31}+z_{41}+z_{32}<4$
	r_2r_7:	$z_{21}+z_{32}<4$		

We are now in the position to establish that RO^4 policy (the conjunction of RO^{RCO}, RO^{RFD}, RO^{RFD^2}, and RO^{ROD}), call it supervisor Δ_4, satisfies Property 2.2. Supervisor Δ_4 is a control policy such that it admits the enabled controllable event α if and only if $\delta(q,a)$ satisfies $RO^{RCO} \wedge RO^{RFD} \wedge RO^{RFD^2} \wedge RO^{ROD}$. Formally, it is stated as follows.

Definition 3.3.5: Supervisor $\Delta_4 = RO^{RCO} \wedge RO^{RFD} \wedge RO^{RFD^2} \wedge RO^{ROD}$.

The intuition behind this control policy is that it ensures that if a shared resource (i.e., a PFD resource) is filled with FD parts, at least one can be advanced out of the shared resources and, thus, out of RCO, which can then operate under RORCO. Furthermore, clearing RCO of this part will not create problems in the FD resources. To summarize, RO^{RFD} allows states with at most one FD resource filled with parts that are FD on the same unreliable resource. RO^{RFD2} allows states for which at most two FD resources are capacitated with FD parts. RO^{ROD} admits states for which at most one resource of ROD is capacitated with FD parts. Wang et al. (2008) establish that Δ_4 is a robust controller for systems where every part type requires at most one unreliable resource.

4. Robust control for product routings with multiple unreliable resources

In Section 3, we develop robust controllers for the single unit resource allocation systems with multiple unreliable resources. These guarantee that if any subset of resources fails, parts in the system requiring failed resources do not block production of parts not requiring failed resources. To establish supervisor correctness, we assume that each part type requires at most one unreliable resource in its route. We now relax this assumption using a central buffer and present robust controllers that guarantee robust operation without assumptions

on route structure. To this end, we will construct new robust controllers in conjunction with the robust controllers, Δ_1 and Δ_2, developed in Subsection 3.1. The following three subsections will demonstrate the way we use a central buffer to extend Δ_1 and Δ_2 for systems where parts may require multiple unreliable resources.

4.1 Route partitioning algorithm

We now show how to use a central buffer to extend Δ_1 and Δ_2 for systems where parts may require multiple unreliable resources. We partition routes with multiple unreliable resources into subroutes, each of which contains one unreliable resource. A part in the last stage of a subroute can move to the first resource of the succeeding subroute or into the central buffer. With this partition, the system resembles one with at most one unreliable resource per route, allowing us to apply Δ_1 and Δ_2.

The route partitioning algorithm (RPA) performs this operation. It starts with the last stage and builds the subroute backwards. A subroute is extended until two unique unreliable resources are detected. Then, a new subroute is begun. We demonstrate below on P_1 of Figure 7.

Route Partitioning Algorithm (RPA)

Algorithm Notation: j, q, u are indices and counters; ε is the empty list; Ψ is a temporary set.

for j=1… $|P|$

let u= $|P_j|$, q=1, $SP_{j1}=\varepsilon$, $\Psi=\varnothing$

while $u\neq 0$

(a) if $\rho(P_{ju})\in R^U\setminus\Psi$, $\Psi=\Psi\cup\{\rho(P_{ju})\}$

(b) if $|\Psi|<2$, $SP_{jq}=push(P_{ju},SP_{jq})$, u=u−1

(Note: The function 'push' takes two parameters, an object and an ordered list of objects, and inserts the object into the head of the list.)

(c) else $\Psi=\varnothing$, q=q+1, $SP_{jq}=\varepsilon$

end while

NS_j = q *(Number of Segments for P_j)*

For j=1, u= $|P_1|$ =8, q=1, $SP_{11}=\varepsilon$, $\Psi=\varnothing$. Then, $\rho(P_{18})=r_1\notin R^U\setminus\Psi=\{r_2,r_4,r_5,r_7\}$, execute (b): $SP_{11}=\langle P_{18}\rangle$, u = 7.

Next, $\rho(P_{17})=r_7\in R^U\setminus\Psi=\{r_2,r_4,r_5,r_7\}$, execute first if: $\Psi=\Psi\cup\{r_7\}=\{r_7\}$. Since $|\Psi|<2$, execute (b) $SP_{11}=\langle P_{17},P_{18}\rangle$, u=6.

Next, $\rho(P_{16})=r_6\notin R^U\setminus\Psi=\{r_2,r_4,r_5\}$, execute (b): $SP_{11}=\langle P_{16},P_{17},P_{18}\rangle$ and u=5.

Next, $\rho(P_{15})=r_5\in R^U\setminus\Psi=\{r_2,r_4,r_5\}$, execute (a): $\Psi=\Psi\cup\{r_5\}=\{r_5,r_7\}$. Since $|\Psi|=2$, execute (c): $\Psi=\varnothing$, q=2, $SP_{12}=\varepsilon$. This completes the first subroute $SP_{11}=\langle P_{16},P_{17},P_{18}\rangle$.

Next, u=5, $\rho(P_{15})=r_5\in R^U\setminus\Psi=\{r_2,r_4,r_5,r_7\}$, execute (a): $\Psi=\Psi\cup\{r_5\}=\{r_5\}$. Since $|\Psi|<2$, execute (b): $SP_{12}=\langle P_{15}\rangle$, u = 4.

Next, $\rho(P_{14})=r_4\in R^U\setminus\Psi=\{r_2,r_4,r_7\}$, execute (a): $\Psi=\Psi\cup\{r_4\}=\{r_4,r_5\}$. Since $|\Psi|=2$, execute (c): $\Psi=\varnothing$, q=3, $SP_{13}=\varepsilon$. This completes the second subroute $SP_{12}=\langle P_{15}\rangle$.

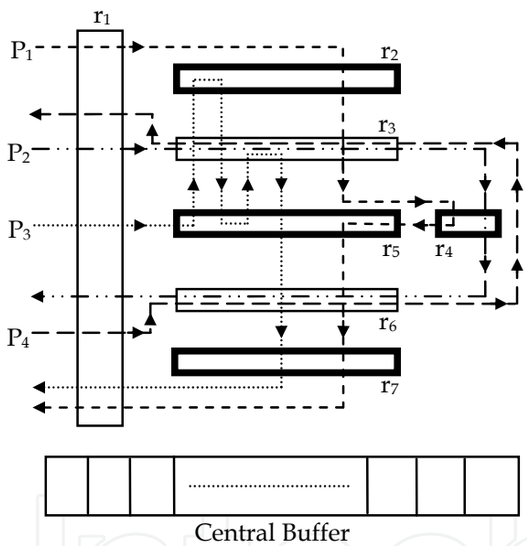
Continuing as shown, RPA partitions P_1 into four subpart types (the remaining two are $SP_{13}=\langle P_{13},P_{14}\rangle$ and $SP_{14}=\langle P_{11}, P_{12}\rangle$) with subroutes $TS_{11}=\langle r_6,r_7,r_8\rangle$, $TS_{12}=\langle r_5\rangle$, $TS_{13}=\langle r_3,r_4\rangle$, and $TS_{14}=\langle r_1,r_2\rangle$. Note that each subroute requires at most one unreliable resource, although the frequency of that resource is not limited. RPA does not affect part types whose routes require at most one unreliable resource.

The maximum number of iterations of the RPA while loop is bounded by the number of part type stages, and thus RPA is no worse than O(CRL=$\sum_{P_j \in P}|P_j|$), which is polynomial in cumulative route length (CRL).

4.2 Central buffer constraints

The central buffer (CB) will be used to clear workstation buffer space of failure-dependent parts that have finished a subroute. If such parts have completely finished their original routes, they exit the system. Otherwise, they must have available space in the CB. This will ensure that they do not block the production of other part types.

For example, suppose the system of Figure 7 is in a state as follows: r_7 is failed with p_{17} waiting for processing; r_5 is holding a completed p_{15}; and r_4 is holding a completed p_{14}. Because of the blocking effect of p_{14} and p_{15}, it is not possible to produce all other part types. However, if we relocate p_{14} and p_{15} to the CB, the system can continue producing P_2, P_3, and P_4. CB constraints are necessary to achieve this. For P_1, we state the linear inequality: $(x_{11}+y_{11})+(x_{12}+x'_{12}+y_{12})+(x_{13}+y_{13})+(x_{14}+x'_{14}+y_{14})+(x_{15}+x'_{15}+y_{15}) \le B_1$, where x_{jk} and y_{jk} are the number of finished and unfinished p_{jk}'s at $\rho(P_{jk})$, x'_{jk} is the number of finished p_{jk}'s relocated to the CB, and B_j the CB space reserved for P_j.



$R=\{r_1,r_2,r_3,r_4,r_5,r_6,r_7\}$ $P=\{P_1,P_2,P_3,P_4\}$
$R^U=\{r_2,r_4,r_5,r_7\}$
$P_1=\{P_{11},P_{12},...,P_{18}\}$ $T_1=\langle r_1,r_2,r_3,r_4,r_5,r_6,r_7,r_1\rangle$
$P_2=\{P_{21},P_{22},...P_{25}\}$ $T_2=\langle r_1,r_3,r_4,r_6,r_1\rangle$
$P_3=\{P_{31},P_{32},...,P_{3,11}\}$ $T_3=\langle r_1,r_5,r_3,r_2,r_3,r_5,r_3,r_5,r_6,r_7,r_1\rangle$
$P_4=\{P_{41},P_{42},P_{43},P_{44}\}$ $T_4=\{r_1,r_6,r_3,r_1\}$
$C=\langle C_1,C_2,...,C_7\rangle=\langle 1,1,1,1,1,1,1\rangle$

Fig. 7. Example with four unreliable resources

With this constraint, finished parts p_{12}, p_{14}, and p_{15}, for subpart types SP_{14}, SP_{13}, and SP_{12}, respectively, can be moved to the CB. Thus, in the example, we can transfer the finished p_{14} and p_{15} to the CB, allowing P_2, P_3, and P_4 to continue production. In the meantime, we decrement x_{14} and x_{15} by 1, and increment x'_{14} and x'_{15} by 1. As an aside, we decrement x'_{14} by 1 and increment y_{15} by 1 when p_{14} advances from the CB into the buffer of r_5.

We now state the CB constraint, CBC. Let $P^*=\{P_j:P_j\in P \wedge |T_j\cap R^U| > 1\}$ be the set of part types that require multiple unreliable resources, and B the total capacity of the CB. For a part type $P_j\in P^*$, let

$$Z_j = \sum_{P_{jk}\in P_j \setminus SP_{j1}} (x_{jk}+y_{jk}) + \sum_{P_{jk}\in LP_j} x'_{jk}$$

where LP_j is the set of "last" part type stages in the subparts of P_j (except SP_{j1}, the final stage of P_j). For example, $LP_1=\{P_{12},P_{14},P_{15}\}$ and $LP_3 = \{P_{32},P_{34},P_{36},P_{38}\}$. In general,

$$LP_j = \{P_{j,|SP_{j,NS_j}|}, P_{j,|SP_{j,NS_j}|+|SP_{j,NS_j-1}|}, \ldots, P_{j,|SP_{j,NS_j}|+\ldots+|SP_{j2}|}\} .$$

Z_j keeps track of the total number of instances of part type stages of $P_j \in P^*$ that are in the system. CBC is defined as:

$$\text{(i)}\ \ Z_j \le B_j,\ \ P_j \in P^* \qquad \text{(ii)}\ \sum_{P_j \in P^*} B_j \le B$$

CBC ensures that every part in the system requiring multiple unreliable resources has capacity reserved on the CB. CBC has no more than $CRL^*|P|$ constraints and thus checking CBC computation is no worse than $O(CRL^*|P|)$, which is polynomial in stable measures of system size.

The level of B_j for $P_j \in P^*$ can be fixed, in which case B_j does not change; or state-based, where we periodically reallocate CB across all $P_j \in P^*$. Although we cannot preempt CB space from parts that have it reserved, we can reallocate CB space that is not reserved. One simple approach is to let $B_j=Z_j$ as long as (ii) holds. This represents a first-come-first-serve rule. Alternatively, we can solve the following assignment problem:

$$\min \sum_{i=1}^{B}\sum_{j=1}^{|P^*|} C_{ij}X_{ij} \tag{1}$$

$$\text{st.} \quad B_j = \sum_{i=1}^{B} X_{ij}, \quad j=1\ldots|P^*| \tag{2}$$

$$Z_j \le \sum_{i=1}^{B} X_{ij}, \quad j=1\ldots|P^*| \tag{3}$$

$$\sum_{i=1}^{B}\sum_{j=1}^{|P^*|} X_{ij} \le B \tag{4}$$

$$X_{ij} \in \{0,1\}, \ i=1\ldots B,\ j=1\ldots|P^*| \tag{5}$$

Here, X_{ij} is 1 if the i^{th} unit of CB is assigned to $P_j \in P^*$, 0 otherwise. The objective (1) minimizes assignment cost; (2) counts the assignment to each $P_j \in P^*$; (3) assures no preemption from parts in the system; and (4) assures the CB is not over allocated. C_{ij} is the cost of assigning CB space to $P_j \in P^*$. This cost could reflect production priorities or failure probabilities. This problem can be solved in polynomial time using the Hungarian Algorithm (Papadimitriou, 1982). The solution frequency is a topic for future research.

4.3 Robust controllers with CBC

We now define two supervisory controllers. The first is the conjunction of Δ_1 and CBC; and the second is the conjunction of Δ_2 and CBC. Recall that Δ_1 and Δ_2 are the controllers of Subsection 3.1. Formally, the extended supervisors are stated as follows.

Definition 4.3.1: Supervisor $\Delta_5 = \Delta_1 \wedge CBC$.
Definition 4.3.2: Supervisor $\Delta_6 = \Delta_2 \wedge CBC$.
The following theorems establish that these supervisors ensure robust operation.
Theorem 4.3.1: Δ_5 is robust to failure of R^U.
Proof: The structure of the proof is as follows. We assume the system to be in an admissible state with parts requiring multiple unreliable resources, with some failed. We show that these parts can advance into the CB or into the buffer space of failure-dependent resources, where they do not block production of parts not requiring failed resources. Let $P_j \in P^*$. The subpart types of P_j constructed by RPA are $\{SP_{j,NS_j}, SP_{j,(NS_j-1)}, \ldots, SP_{j1}\}$. Assume that in the current state, q, unreliable resources in the subroutes of P_j have failed and that q satisfies Δ_5.
In the following, we want to show that under Δ_5 parts of type P_j do not block other part types from producing. We ignore parts of type P_j in the final subroute since it is covered by Δ_1. That is, Δ_1 guarantees that parts in the final subroute can be advanced into the buffer space of the last resource and completed and removed from the system if the resource is operational or stored there, out of the way of part types not requiring failed resources, if it is not.
Let $\zeta_{qj}=\{p_{jk} \mid P_{jk} \in SP_{jq}, q = NS_j, (NS_j-1),\ldots,2\}$ be the set of parts of P_j in the state q. Let $\wp_{qj}=\{p_{jk} \mid P_{jk} \in LP_j\}$ be the set of parts of P_j in the final stage of a subroute. By the definition of LP_j, $\wp_{qj} \subseteq \zeta_{qj}$. Now, Δ_1 guarantees that all parts in $\zeta_{qj} \setminus \wp_{qj}$ can be advanced, perhaps through several processing steps, into the buffer spaces of resources required by stages of LP_j. That is, Δ_1 guarantees a sequence of part movements such that the system reaches a new state, say t, where $\zeta_{tj}=\wp_{tj}$. In state t, all instances of P_j are at the end of a subroute.
The left hand side of CBC does not change in moving from state q to state t. To see this, note that CBC is only affected by parts in P*. Since we allow no new parts to be admitted and no part of P* is required to move from one subroute to another (only to the end of the current subroute), the left-hand-side of CBC does not change magnitude. Thus, the part advancement under Δ_1 does not violate CBC.Now, CBC guarantees that every part of ζ_{tj} has capacity reserved on the CB, and any finished part of this set can be moved to the CB. Further, any unfinished part of ζ_{tj} can be finished and moved to the CB if its resource is operational. If the associated resource is not operational, the part can be stored at its failed resource where it will not block the production of part types not requiring failed resources. Thus, all operational resources can be cleared of parts of type P_j. Under Δ_1, the resulting state is a feasible initial state if resource repairs or additional failures occur.
Theorem 4.3.2: Δ_6 is robust to failure of R^U.
Proof: The proof follows the same construction as Theorem 4.3.1. The main difference is in how BA and SSLA operate. Thus, Δ_5 and Δ_6 guarantee robust operation for systems where parts can require multiple unreliable resources. Note that if every resource is unreliable, both theorems continue to hold.

5. Conclusion and future research

Supervisory control for manufacturing systems resource allocation has been an active area of research. Significant amount of theories and algorithms have been developed to allocate resources effectively and efficiently, and to guarantee important system properties, such as system liveness, traceability, deadlock-free operations. However, a major assumption these research works are based on is that resources never fail. While resource failures in automated

manufacturing systems are inevitable, we investigate such system behaviours and control dynamics. First, we developed the notion of robust supervisory control for automated manufacturing systems with unreliable resources. Our objective is to allocate system buffer space so that when an unreliable resource fails the system can continue to produce all part types not requiring the failed resource. We established properties that such a controller must satisfy, namely, that it ensure safety for the system given no resource failure; that it constrain the system to feasible initial states in case of resource failure; that it ensure safety for the system while the unreliable resource is failed; and that during resource repair it constrain the system to states that will be feasible initial states when the repair is completed.
We then developed a variety of control policies that satisfy these robust properties.

Taxonomy for Future Research Directions		
System Structure	S1	at most one unreliable resource for each part type
	S2	random number of unreliable resources for each part type
Central Buffer Capacity	C1	without central buffer
	C2	with central buffer
Flexible Routing	FR1	every part type stage can be performed by exactly one resource
	FR2	every part type stage can be performed by exactly two resources
	…	
	FRj	every part type stage can be performed by exactly j resources
Robustness Level	RB1	no resource failures
	RB2	at most one resource failure at any time
	RB3	at most two resource failures at any time
	…	
	RBi	at most i resource failures at any time
Unreliable Resource Condition	RC1	unreliable resources fail at any time
	RC2	unreliable resource failure characteristics can be estimated
Application Areas	AA1	Manufacturing Systems
	AA2	Business Processes and Workflow Management
	AA3	E-Commerce
	AA4	Supply Chain Management
	AA5	Internet Resource Mangement
	AA6	Transporation Systems
	AA7	Healthcare Systems

Table 1. Taxonomy for future research directions

Specifically, supervisory controllers Δ_1-Δ_4 are for systems with multiple unreliable resources where each part type requires at most one unreliable resource. Supervisory controllers Δ_5-Δ_6 control systems for which part types may require multiple unreliable resources. Another classification of the controllers is based on the underlying control mechanism: controllers Δ_1-Δ_3 'absorb' all parts requiring failed resources into the buffer space of failure-dependent

resources, controller Δ_4 distribute' parts requiring failed resources among the buffer space of shared resources, and controllers Δ_5-Δ_6 utilize central buffer to achieve robust operations. These robust controllers assure different levels of robust system operation and impose very different operating dynamics on the system, thus affecting system performance in different ways. An extensive simulation study has been conducted and a set of implementation guidelines for choosing the best robust controller based on manufacturing system characteristics and performance objectives are developed in Wang et al. (2009).

A taxonomy is developed and presented in Table 1 to help guide future research in the area of robust supervisory control. By combining the different system structures, the presence/absence of central buffer, flexible routing capability, system robust level requirements, and unreliable resource failure characteristics, a significant amount of future research and development need to be done to address a variety of system control and performance requirements. And, although automated manufacturing systems are the context in which we develop the robust supervisory control research. We expect to expand our research to other application areas due to the similarity in resource allocation requirement and complexity in workflow management. The robust controllers we developed so far only address a small subset of the research taxonomy. For example, controller Δ_1 falls in the category in the taxonomy of (S1, C1, FR1, RB2, RC1, AA1). Especially, it would be interesting and challenging to develop supervisory control policies for systems with flexible routing and for systems where the failure characteristics of resources are dynamically evolving and can be estimated through sensor monitoring and degradation modelling.

6. References

Chew, S. & Lawley, M. (2006). Robust Supervisory Control for Production Systems with Multiple Resource Failures. *IEEE Transactions on Automation Science and Engineering,* Vol.3, No.3, (July 2006), pp. 309-323, ISSN 1545-5955

Chew, S.; Wang, S. & Lawley, M. (2008). Robust Supervisory Control for Product Routings with Multiple Unreliable Resources. *IEEE Transactions on Automation Science and Engineering,* Vol.6, No.1, (January 2009), pp. 195-200, ISSN 1545-5955

Chew, S.; Wang, S. & Lawley, M. (2011). Resource Failure and Blockage Control for Production Systems. *International Journal of Computer Integrated Manufacturing,* Vol.24, No.3, (March 2011), pp. 229-241, ISSN 0951-192X

Cormen, T.; Leiserson, C. & Rivest, R. (2002). *Introduction to Algorithms* (Second Edition), McGraw-Hill, ISBN 0072970545, New York, USA

Ezpeleta, J.; Tricas, F.; Garcia-Valles, F. & Colom, J. (2002). A Banker's Solution for Deadlock Avoidance in FMS with Flexible Routing and Multiresource States. *IEEE Transactions on Robotics and Automation*, Vol.18, No.4, (August 2002), pp. 621–625, ISSN 1042-296X

Habermann, A. (1969). Prevention of System Deadlocks. *Communications of the ACM*, Vol.12, No.7, (July 1969), pp. 373–377, ISSN 0001-0782

Hsieh, F. (2004). Fault-tolerant Deadlock Avoidance Algorithm for Assembly Processes. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, Vol.34, No.1, (January 2004), pp. 65-79, ISSN 1083-4427

Lawley, M. (1999). Deadlock Avoidance for Production Systems with Flexible Routing. *IEEE Transactions on Robotics and Automation*, Vol.15, No.3, (June 1999), pp. 497-510, ISSN 1042-296X

Lawley, M. (2002). Control of Deadlock and Blocking for Production Systems with Unreliable Resources. *International Journal of Production Research,* Vol.40, No.17, (November 2002), pp. 4563-4582, ISSN 0020-7543

Lawley, M. & Reveliotis, S. (2001). Deadlock Avoidance for Sequential Resource Allocation Systems: Hard and Easy Cases. *International Journal of Flexible Manufacturing Systems,* Vol.13, No.4, (October 2001), pp. 385-404, ISSN 0920-6299

Lawley, M.; Reveliotis, S. & Ferreira, P. (1998). Application and Evaluation of Banker's Algorithm for Deadlock-free Buffer Space Allocation in Flexible Manufacturing Systems. *International Journal of Flexible Manufacturing Systems,* Vol.10, No.1, (February 1998), pp. 73–100, ISSN 0920-6299

Lawley, M. & Sulistyono, W. (2002). Robust Supervisory Control Policies for Manufacturing Systems with Unreliable Resources. *IEEE Transactions on Robotics and Automation*, Vol.18, No.3, (June 2002), pp. 346-359, ISSN 1042-296X

Papadimitriou, C. (1982). *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, ISBN 0486402584, New Jersey, USA

Park, S. & Lim, J. (1999). Fault-tolerant Robust Supervisor for Discrete Event Systems with Model Uncertainty and Its Application to a Workcell. *IEEE Transactions on Robotics and Automation*, Vol.15, No.2, (April 1999), pp. 386–391, ISSN 1042-296X

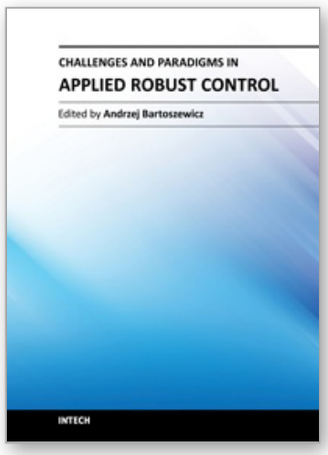
Ramadge, P. & Wonham, W. (1987). Supervisory Control of a Class of Discrete Event Processes. S*IAM Journal on Control and Optimization,* Vol.25, No.1, (March 1985), pp. 206–230, ISSN 0363-0129

Reveliotis, S. (1999). Accommodating FMS Operational Contingencies through Routing Flexibility. *IEEE Transactions on Robotics and Automation*, Vol.15, No.1, (February 1999), pp. 3–19, ISSN 1042-296X

Reveliotis, S. (2000). Conflict Resolution in AGV Systems. *IIE Transactions*, Vol.32, No.7, (July 2000), pp. 647-659, ISSN 0740-817X

Wang, S.; Chew, S. & Lawley, M. (2008). Using Shared-Resource Capacity for Robust Control of Failure-Prone Manufacturing Systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, Vol.38, No.3, (May 2008), pp. 605-627, ISSN 1083-4427

Wang, S.; Chew, S. & Lawley, M. (2009). Guidelines for Implementing Robust Supervisors in Flexible Manufacturing Systems. *International Journal of Production Research,* Vol.47, No.23, (December 2009), pp. 6499-6524, ISSN 0020-7543



Challenges and Paradigms in Applied Robust Control

Edited by Prof. Andrzej Bartoszewicz

ISBN 978-953-307-338-5
Hard cover, 460 pages
Publisher InTech
Published online 16, November, 2011
Published in print edition November, 2011

The main objective of this book is to present important challenges and paradigms in the field of applied robust control design and implementation. Book contains a broad range of well worked out, recent application studies which include but are not limited to H-infinity, sliding mode, robust PID and fault tolerant based control systems. The contributions enrich the current state of the art, and encourage new applications of robust control techniques in various engineering and non-engineering systems.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Shengyong Wang, Song Foh Chew and Mark Lawley (2011). Robust Control for Single Unit Resource Allocation Systems, Challenges and Paradigms in Applied Robust Control, Prof. Andrzej Bartoszewicz (Ed.), ISBN: 978-953-307-338-5, InTech, Available from: http://www.intechopen.com/books/challenges-and-paradigms-in-applied-robust-control/robust-control-for-single-unit-resource-allocation-systems

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen