Accommodating FMS Operational Contingencies through Routing Flexibility

Spyros A. Reveliotis

School of Industrial & Systems Engineering
Georgia Institute of Technology
765 Ferst Drive
Atlanta, GA 30332

Abstract

Overwhelmed by the complexity of the FMS Deadlock Avoidance problem, current research has, for the most part, ignored the aspects (and benefits) related to flexible (dynamic) job routing. Extending current structural control policies, based on static job routing, to accommodate routing flexibility is nontrivial, primarily due to the fact that the possible routing options for a single job can grow exponentially fast. Hence, computationally efficient techniques are required to incorporate the inherent FMS routing flexibility to current structural control schemes. This paper undertakes the investigation of the problem of integrating routing flexibility in FMS structural control, by addressing the problem of "optimal" job re-routing in case of operational contingencies. Analytical formulations and efficient solution algorithms are developed for the case that the FMS is structurally controlled by a class of recently emerging polynomial-complexity, one-step lookahead deadlock avoidance policies, collectively known as Ω DAP's.

Keywords: Routing Flexibility, Deadlock Avoidance, FMS Real-Time Control, Contingency Accommodation

KEY NOTATION

$\mathcal{R} = \{R_1, \ldots, R_m\}$: system resources

$\mathcal{J} = \{JT_1, \ldots, JT_n\}$: supported job types

JT_{jk}: k-th processing stage of the j-th job type

$\mathcal{R}_{jk} = \{R^l_{jk}\}$: the set of resources able to support the processing of stage JT_{jk}

$\mathcal{JR}_j = \mathcal{R}_{j1} \times \ldots \times \mathcal{R}_{j,l(j)}$: set of feasible routes for job type JT_j

m: number of resource types

$\bar{C}$: maximal resource capacity

D: total number of distinct job stages

$\bar{l}$: maximal length of job type sequences

K: largest possible number of options supporting a single job stage

$\mathcal{S} = \{s^i\}$: set of RAS states

$\mathcal{S}_r(\mathcal{P})$: reachable subspace under some correct deadlock avoidance policy P.

$\mathbf{s}(t)$: RAS state at time instant t

$\mathcal{I} = \{j_j\}$: set of job instances to be re-routed because of a contingency occurrence

w_j: the significance weight of job instance j_j

1 Introduction

FMS Supervisory Control and Deadlock Avoidance In the emerging paradigm of *agile* manufacturing, it has become clear that: (i) effectively exploiting the inherent flexibility(-ies) of contemporary manufacturing systems is of paramount importance for supporting the new manufacturing strategies (e.g., [1, 2]), and (ii) the effective management of this flexibility requires the rigorous modeling and analysis of the system behavior by the application of appropriate control-theoretic techniques (e.g., [3, 4]). As a result, a new area in the real-time control of flexibly automated manufacturing systems has been developed, broadly known as *Structural or Supervisory* control of these environments [5, 6]. The main theme of this area is the modeling of the (discrete-part) manufacturing system as a *Discrete-Event System (DES)*, and the confinement of its behavior – defined by the realizable event sequences – so that it is consistent with the broader production objectives.

Currently, the issue most extensively studied in the FMS supervisory control context is that of establishing *nonblocking* (or *deadlock-free*) behavior. In the broader context of resource allocation systems (RAS), *deadlock* arises when a subset of executing processes is entangled in a circular-waiting situation, where each of these processes, in order to proceed, requires the acquisition of some resource(s) presently held by another process in the subset [7]. In a manufacturing system setting, the primary resource giving rise to deadlock is the *finite buffering capacity* of the system equipment (i.e., processing stations and material handling system). Specifically, each job in the system, being a physical entity, always requires one unit of buffering capacity to be staged in, which it releases only upon being transferred to another unit.

A series of interesting results on the problem of FMS deadlock-free operation are reported in [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. All these works perceive the manufacturing system as a *resource allocation system* shared by concurrently executing *processes*. Each process is defined as a sequence of *processing stages*, with each stage further characterized by the set of resources required for its execution. Hence, formally, a process is described as a sequence of *resource allocation vectors*. In the special case where the only resource at issue is the buffering capacity of the system entities, this process sequence essentially defines the part *route* through the system. A formal classification of discrete-part manufacturing systems based on the structure of the resource allocation vectors describing their admissible process stage requirements, is provided in [17]. The main classes identified are: (i) the *Single-Unit* RAS, in which every process at each stage requires a single unit of a single resource (i.e., a unit of buffering capacity), (ii) the *Single-Type* RAS, where every process stage requires an arbitrary number of units of a single resource type (modeling, thus, jobs organized into tightly connected batches), (iii) the *Conjunctive (AND)* RAS, where every processing stage requires an arbitrary number of units from an arbitrary number of resource types (i.e., beyond the required buffering capacity, jobs might also contest for some scarce auxiliary resources, like pallets and fixtures), and finally, (iv) the *Disjunctive–Conjunctive (OR/AND)* RAS, where every stage is associated with a set of AND-type resource allocation vectors, the implication being that allocation of any of these requests is adequate for the execution of this stage (hence, this RAS model captures also the routing flexibility inherent in contemporary FMS).

In [17] it is also shown that higher classes of the above taxonomy subsume the lower-class models, and the complexity of the underlying control problem is strongly dependent on the RAS category. Given that the problem complexity is significantly high even in its simplest formulation (i.e., Single-Unit RAS), most of the current research has essentially focused on the first three classes of the above taxonomy, ignoring the effects of routing flexibility in the policy development.[1] Hence, it is typical in the existing literature that a *static* routing scheme is assumed, where the entire routing of a job through the system is predetermined upon its loading. This fixing of the future job resource requirements facilitates the analysis of the evolution of the system resource allocation status. As it will be shown in the next section, the generic disjunctive scheme, in which every job stage can be supported by a number of different resources, leads to a number of routes which is exponential to the job route length, and any (naive) approach to the problem can suffer from exponential space complexity.

FMS Operational Flexibilities The flexibility implemented in modern manufacturing systems also constitutes the topic of a significant body of the current manufacturing systems literature. However, most of this work addresses the issue at a conceptual level, attempting to outline (i) the different manifestations of this flexibility in the system behavior and performance, and (ii) the ways in which it is supported by the system operational features and policies. Some interesting and indicative work along these lines is that presented in, e.g., [23, 24, 25]. Analytical work in the area attempts to justify flexible manufacturing automation by quantifying the trade-offs of a flexible vs. an inflexible environment, from a strategic/economic perspective (e.g., [26, 27, 28, 29]). Recently, some field studies have been reported which attempt to statistically evaluate the extent in which the current manufacturing technology and practices support the advantages attributed to flexibly automated manufacturing systems [4, 30]. A recurrent theme of these studies is that most current implementations of these systems end up being

[1] Some work that has explicitly incorporated dynamic routing in its modeling is presented in [13, 14], and more recently in [22].

operated in a very inflexible manner due to the lack of a thorough understanding of the system dynamics, and consequently, the lack of effective and efficient tactical and operational policies. This result is corroborated by the apparent lack of scientific literature addressing the issue of manufacturing flexibility at the planning (tactical) and operational levels. A recent attempt to capture the effects of various FMS operational flexibilities in models pertaining to the tactical planning problem of discrete-part manufacturing, is the work presented in [31].

Problem Definition Given (i) the "maturity" of the current research with respect to the "classical" FMS deadlock avoidance problem, and (ii) the broadly admitted significance of flexible dynamic routing for enhancing the system robustness and performance, in this paper we consider *the integration of deadlock-avoidance control with dynamic policy reconfiguration and job re-routing, in order to accommodate / minimize the disruptive effects of operational contingencies like (i) machine breakdowns and (ii) the arrival of expedient ("hot") jobs*. The implementation of this idea, and the detailed problem formulation strongly depend on the control paradigm used for deadlock avoidance. The paradigm employed in this work is that of Ω-Deadlock Avoidance Policies (Ω DAP's), applied in Single-Unit RAS [15, 16]. Ω DAP's are essentially one-step lookahead policies, which allow a tentative job stage transition if and only if the resulting resource allocation state presents some policy-defining property, which guarantees deadlock and restricted-deadlock-free operation.[2] An additional property that makes them amenable to the application context considered in this work, is that the computational cost for their initial set-up and execution is polynomial with respect to the system size (i.e., distinct resource types and job stages). Hence, they are easily *reconfigurable*, in case of an operational contingency.

The specific Ω DAP's considered in this work are RUN [16], RO [18, 32], Banker's [19], and the Optimal policy [17].[3] The first two of these policies are expressed as a set of algebraic constraints (linear inequalities) on the system resource allocation state, while the others are established on conditions verified through polynomial search-based algorithms. It turns out that the detailed formulation and the solution techniques for the problem proposed herein are essentially different for each of these two policy-defining structures. Hence, each case is addressed in a separate part of the paper.

Finally, we would like to emphasize that the contingency accommodation problem undertaken in this paper is different from the typical deadlock avoidance problem under dynamic job

[2] A more formal characterization of the system operation and the detailed definition of the deadlock-related concepts, are provided in the next section.

[3] Notice that the Optimal DAP is polynomially implementable only for a sub-class of Single Unit-RAS [17]

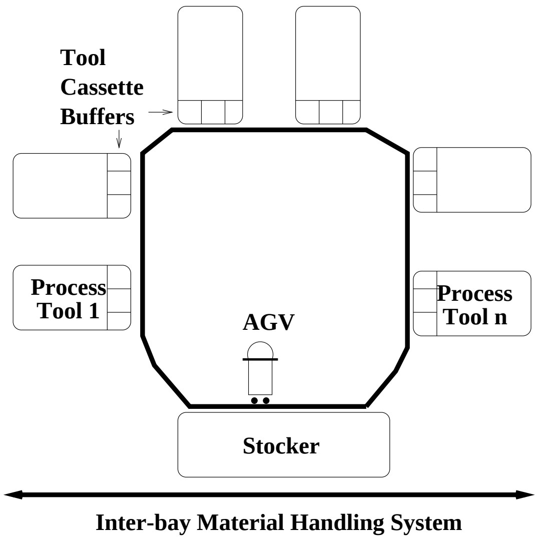

Figure 1: A typical lay-out for a semiconductor fab bay

routing. More specifically, while the latter attempts to re-distribute the system workload at the different workstations in order to maintain a balanced machine load and improve performance, the former seeks to minimize the disruption caused by the occurrence of unexpected events that drastically change the system structure. Hence, the need for policy reconfiguration and job re-routing in case of a machine breakdown, will arise even in an FMS controlled by one of the deadlock avoidance schemes presented in [13] or [14], which allow dynamic job routing. Furthermore, the appropriateness of some policy for one of these problems/environments does not immediately imply its appropriateness for the other. As an example, the policies presented in [13, 14, 22] although amenable to dynamic job routing, might face considerable difficulty with accommodating operational contingencies like machine break-downs, since their *reconfigurability* is rather limited, due to the fact that their deployment requires the enumeration of the *siphons* in the underlying Petri Net model, which is an operation of non-polynomial computational cost [33].

An Example Application To provide more concrete context for the above problem description, consider the operation of the semiconductor manufacturing fab. According to [34], as the industry moves from 200-mm to 300-mm wafer diameters, physical and economical considerations render the automation of the wafer handling an imperative effort. Of particular interest to this work are the material-handling operations at the, so called, *intra-bay* level.

The organization of a fab bay unit is presented in Figure 1. Its main elements are: (i) a *stocker* unit, where cassettes of wafers requiring the bay processes are kept stored until their loading and removal from the bay, (ii) a number of *process tools*, which constitute the bay's main processing units, and (iii) an interconnecting material handling system transferring wafer cassettes among the process tools and the stocker. Typically, the material handling system is a single-loop Automated Guided Vehicle (AGV) or Track Guided Vehicle (TGV) system, and therefore, its control with respect to deadlock-free material flow is a rather trivial task: this unit should essentially act as the facilitator of the cassette transport between the different process tools and/or the stocker unit, as commanded by the *material control system*. However, controlling the flow at the process tool level is a complicated task. Although in the past fabs have been organized according to a functional lay-out, grouping process tools corresponding to the same process in a single bay unit, currently there is a tendency to develop bays as *mini-environments* which integrate a number of different process types, since such an approach leads to reduced cycle times, higher levels of cleanness, and therefore, better yields. As a result, it is typical that a cassette will visit a number of chambers / processes before it returns to the bay stocker. Furthermore, the demand for increased levels of flexibility, combined with the lengthy processing times of the underlying operations, require the concurrent processing of cassettes with wafers of different types. Hence, the routing of different cassettes through the processes of a given bay may be quite arbitrary. On the other hand, economic and operational/space considerations impose rather strict limits to the buffering capacity of the process tools, in terms of cassettes. According to [34], most typically, each process tool can hold at most four wafer cassettes. Hence, it should be obvious that such an environment is prone to deadlock, and any successful deployment requires the systematic treatment of the problem through some deadlock resolution strategy.

Furthermore, the process tools of a fab bay present significant redundancy in terms of their processing capabilities [35]. For instance, it might be the case that two or more photo(-lithography) machines have essentially identical capabilities, but still be differentiated on the basis of auxiliary equipment like process masks. In other cases, even though two or more machines can perform the same set of operations, some of them are more efficient than other with respect to certain operation subsets, in terms of processing times and yields. Therefore, it is customary to discriminate between *primary* and *secondary* alternatives for the different operations supported by the bay equipment.

Additionally, contemporary fab process tools are quite prone to failures, not so much because of mechanical breakdown, as because of decalibration. Experienced downtimes are quite lengthy, ranging in the order of several hours [35]. Finally, the extensive variability in fab operations, and the sensitive nature of the undergoing processes, make the arrival of *"hot"* jobs at the fab bays a not so unusual phenomenon.

It is clear from the above description, that material control at the fab intra-bay level presents all the basic features defining the problem addressed by this paper. Furthermore, the availability of a decision tool able to react to the major disruptions occurring during the fab bay operation in a systematic and rational way, contributes to the higher levels of productivity, responsiveness and yield sought by these environments.

Paper Overview The detailed organization of this paper is as follows: In the next section we provide the required background on the problem of FMS deadlock avoidance, the complexity of its optimal solution, and the basic theory of Ω-DAP's. In Section 3, we formulate and solve the problem of optimally rerouting currently running jobs due to operational contingencies, for the case that the FMS is structurally controlled by algebraically expressed Ω DAP's; two recently developed policies, RUN [16] and RO [18, 32], are considered. Section 4 addresses the contingency accommodation and job rerouting problem in case that the FMS is controlled by (polynomial) search-based one-step lookahead policies. The discussion of this section is developed around a modified version of Banker's algorithm appropriate for the FMS context [19], but the results are generalized to other lookahead schemes, like polynomial-complexity implementations of the optimal DAP – for those sub-classes of Single-Unit RAS that such an implementation is possible [17]. Finally, Section 5 concludes the paper discussion and proposes issues for further research.

2 Preliminaries

The FMS Resource Allocation Model: For the purposes of the analysis pursued in this work, the FMS is considered as a *Resource Allocation System (RAS)*, i.e., a system of *resource types*, $\mathcal{R} = \{R_i,\ i = 1,\ldots,m\}$, with each resource type, R_i, being a pool of a finite number, C_i, of identical units of that type. C_i will be referred to as the *capacity* of resource R_i. Essentially, each of these resources models the finite buffering capacity of one of the system workstations and/or material handling components; in the fab intra-bay automation context, each resource R_i corresponds to a process tool, while C_i defines its cassette buffering capacity. Once configured for a certain production run, the system can support the execution of a set of *job types* $\mathcal{J} = \{JT_j,\ j = 1,\ldots,n\}$, with job type JT_j being defined as a sequence of *stages*, $< JT_{jk},\ k = 1,\ldots,l(j) >$. Each job stage can be executed on any resource from a subset $\mathcal{R}_{jk}$ associated with it, requiring one unit of capacity of the supporting resource during its execution. In the context of the fab bay application, presented in the Introduction, job types are defined by the process plans (*process logs* [35]) describing the processing of a certain wafer type through the bay equipment, during a single visit.

The entire set of routes according to which a job instance of type JT_j can be executed through the system is given by the product set:

$$\mathcal{JR}_j = \mathcal{R}_{j1} \times \mathcal{R}_{j2} \times \ldots \times \mathcal{R}_{j,l(j)} \tag{1}$$

Obviously, the number of possible routes for a certain job type, JT_j, is: $|\mathcal{R}_{j1}| \cdot |\mathcal{R}_{j2}| \ldots |\mathcal{R}_{j,l(j)}|$. This number is $O(K^{\bar{l}})$, where K is an upper bound on the number of possible options for a certain stage, and $\bar{l}$ is an upper bound on the job route length. Hence, *the number of possible routes for a certain job presents exponential space complexity.*

This combinatorial explosion of the possible routing options resulting from the dynamic job routing, has made researchers of the FMS structural control problem ignore this system capability, focusing on situations where each job type is deterministically associated with a certain job route[4]. From a mathematical perspective, this is modeled by requiring that:

$$\forall\ j, k,\ \ |\mathcal{R}_{jk}| = 1 \tag{2}$$

Equation 2 implies that the sequence of the processing requirements of a certain job instance is perfectly known upon its loading into the system. Under this assumption, the system operation is further modeled by a *Finite State Automaton (FSA)* [16], where: An *event* $e \in \mathcal{E}$ of this FSA corresponds to the advancement of any job in the system by one step. The FMS *state* $s^i \in \mathcal{S}$ is defined by the distribution of the currently running jobs in the system to the various processing stages. The automaton *state transition function* $f : \mathcal{S} \times \mathcal{E} \rightarrow \mathcal{S}$ is a formal expression of the specifications for the RAS operation: $f(s^i, e)$ is mapped to the resulting state s^j if the job step defined by event e is *feasible* under the resource allocation described by state s^i; o.w., it is mapped back to state s^i. The *initial* and *final* states of the automaton correspond to state s^0, i.e., the state where the system is idle and empty of any jobs; hence, the *language* accepted by this automaton corresponds to complete production runs. The structure of the state space of this FSA is expressed graphically by the *State Transition Diagram (STD)*, a graph with *nodes* corresponding to the FSA states, and *arcs* corresponding to the feasible state transitions.

[4] or the slightly broader scheme in which each job type can be materialized in a polynomial number of preselected routes, with a job instance being assigned to a route upon its loading into the system – hence, for the deadlock avoidance purposes, each job route essentially defines a (refined) job type. Some more dynamic routing schemes have been recently studied in [13, 14, 22].

FMS Deadlock Avoidance: In this FSA framework, *deadlocks* are identified by *strongly connected* components of the system STD, which do not contain the empty state s^0. By this token, a *correct Deadlock Avoidance Policy (DAP)*, $\mathcal{P}$, tries to restrict the system operation to *a strongly connected component of the system state space which contains the empty state* s^0; this component will be characterized as the subspace $\mathcal{S}_r(\mathcal{P})$ *admissible* by the policy. Given an FMS configuration, an applied DAP is *optimal* if its admissible subspace is the *maximal* subspace presenting the defining property of a correct DAP.

Ω **DAP's:** It is easy to see that under the above assumptions, the optimal DAP, $\mathcal{P}^*$, is uniquely defined for a given FMS configuration. The admissibility of a reachable state s^i by the optimal DAP is based on the existence of an event sequence σ such that $s^i \xrightarrow{\sigma} s^0$; this decision problem is known as the *state safety* problem. However, it has been shown that for the considered class – i.e., deterministically routed Single-Unit RAS – the state-safety problem is NP-complete [36].[5] In view of the nontractability of the optimal DAP, Ω DAP's have been proposed in [15, 16] as a class of correct, scalable and reconfigurable DAP's, appropriate for real-time FMS structural control. These policies are based on the identification of pertinent polynomially computable conditions on the RAS states, such that the resulting admissible subspaces present the DAP correctness-defining property (i.e., strong connectivity with inclusion of the initial state s^0).

As it was mentioned in the Introduction, it is pertinent for the development of this paper, to classify Ω DAP's on the basis of whether their policy-defining condition admits an algebraic (closed-form) characterization or not. The policies with algebraic representation considered in this paper are (i) the *Resource Upstream Neighborhood (RUN)* policy [16] and (ii) the *Resource Ordering (RO)* policy [18, 32]. The policy description, together with the formulation and solution techniques for the optimal job rerouting problem in their context, are provided in the next section. Policies which do not possess a convenient closed-form representation of their admissible state space, but provide efficient (polynomial) search-based one-step lookahead recognition algorithms, are (i) a modified version of Banker's algorithm [19] and (ii) the Optimal DAP, *whenever the set of unsafe states coincides with that of the system deadlock states* [17]. These policies are treated in Section 4.

[5] Practically interesting subclasses for which the optimal DAP is polynomially computable through one-step lookahead are provided in [17, 21].

3 Optimal Job Rerouting for Algebraic Ω DAP's

In this section we consider the problem of optimally rerouting running jobs in the face of operational contingencies, for the case where the FMS is controlled by algebraic Ω DAP's, specifically, RUN [16] or RO [18, 32] DAP's. We proceed with a brief description of the defining logic and the algebraic characterization for each of these policies, as it applies in the Single-Unit RAS context. These characterizations subsequently allow the unified formulation of the optimal job rerouting problem, and the development of efficient solution techniques for it. The discussion is elucidated by two small but highlighting examples.

3.1 The defining logic for RUN and RO DAP's

RUN RUN is an acronym for *Resource Upstream Neighborhood*. The policy draws its name from the underlying idea that higher-capacitated resources should be able to function as buffers for jobs staged in or heading towards them. Hence, the policy associates with each resource an *upstream neighborhood*, i.e., a set of job stages defined according to the following rule:

Definition 1 *[16] The* upstream neighborhood *of resource R_i consists of all route stages JT_{jk} which are supported by resource R_i, plus all the route stages belonging to the maximal route subsequences immediately preceding each of the aforementioned JT_{jk}, and involving stages JT_{jp} with $C_{R(JT_{jp})} \leq C_i$. A job instance j_j is in the neighborhood of resource R_i iff its current processing stage is in the neighborhood of R_i.*

RUN is formally defined as follows:

Definition 2 *[16] A resource allocation state* $\mathbf{s}$ *is accepted by RUN DAP iff the number of jobs in the upstream neighborhood of each resource, R_i, does not exceed its buffering capacity, C_i.*

Example 1 We highlight the policy-defining logic and the resource neighborhood construction through an example. Consider a small system depicted in Figure 2. This system consists of four resources R_1, R_2, R_3, R_4, with a corresponding capacity vector $\mathbf{C} =< 2, 1, 1, 1 >$. Alternatively, one can think of it as a *cluster tool*, consisting of four *chambers*, R_1, R_2, R_3, R_4, each able to hold C_i, $i = 1, \ldots, 4$, cassettes at a time. [6] In its current configuration, the system

[6] An actual fab bay will contain a larger number of process tools, and will also support a larger variety of process types. It is imperative, therefore, that the proposed solutions are computationally efficient. This requirement is attended by the complexity considerations and analysis of the proposed computational schemes. For expository reasons, however, we have maintained the size of the examples discussed in this paper relatively small.

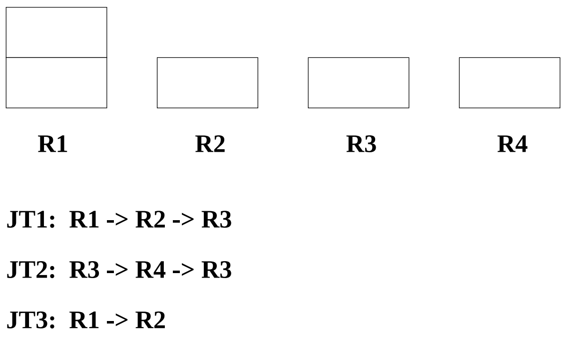

Figure 2: An example Single-Unit RAS

supports the production of three distinct job types, with *primary* job routes: $JT_1 : \ R_1 \rightarrow R_2 \rightarrow R_3$, $JT_2 : \ R_3 \rightarrow R_4 \rightarrow R_3$, $JT_3 : \ R_1 \rightarrow R_2$. By applying the logic of Definition 1 to this system, we obtain the neighborhood inclusions indicated by the following incidence matrix:

$$A_{RUN} = \begin{bmatrix} 1 & & & & & & 1 & \\ & 1 & & & & & & 1 \\ & 1 & 1 & 1 & 1 & 1 & & \\ & & & 1 & 1 & & & \end{bmatrix} \tag{3}$$

Each row of the above matrix corresponds to a resource neighborhood, $N(R_1), \ldots, N(R_4)$. Each column corresponds to a route stage, starting with the stages of job type JT_1, and concatenating the stages of JT_2 and JT_3. Using the definition of the system (FSA) *state*, at time t, provided in Section 2 – i.e., a vector $\mathbf{s}(t)$ with components corresponding to the distinct job stages $JT_{jk}, \ j = 1, \ldots, 3, \ k = 1, \ldots, l(j)$, and with $\mathbf{s}_{jk}(t)$ being equal to the number of job instances executing stage JT_{jk} at time t – it is easy to see that the policy constraints can be expressed by a system of linear inequalities on the system state:

$$A_{RUN}\mathbf{s}(t) \leq \mathbf{C} \tag{4}$$

□

RO RO stands for *Resource Ordering*, and underlies the fact that the policy imposes a total ordering on the system resources, and subsequently it uses this ordering to ensure that there is a free passage for at least one of the classes of jobs moving up or down the order. The complete policy definition is as follows:

Definition 3 *[18, 32]*

1. *Impose a total ordering on the set of system resources $\mathcal{R}$, i.e., a mapping*

$$o \ : \ \mathcal{R} \rightarrow \{1, \ldots, |\mathcal{R}|\} \ s.t. \ R_i < R_j \Leftrightarrow o(R_i) < o(R_j)$$

   We say that "R_i (R_j) is to the left (right) of R_j (R_i)", iff $R_i < R_j$.

   Furthermore, job stage JT_{jk} is characterized as right (left)*-directed if $R(JT_{j(x-1)}) < (>) \ R(JT_{jx})$, $\forall x > k$, where $R(JT_{jk})$ denotes the resource supporting stage JT_{jk}. A stage which is neither right nor left-directed is an* undirected *stage.*

   A job instance is characterized as right, left, or undirected on the basis of its running processing stage.

2. *Let*

   - $RC_i(t) = \{$*right-directed + undirected job instances in* R_i *at time* $t\}$
   - $LC_i(t) = \{$*left-directed + undirected job instances in* R_i *at time* $t\}$

3. *A resource allocation state* $\mathbf{s}(t)$ *is accepted by RO iff*

$$\forall i, j \ : \ R_i < R_j \Rightarrow |RC_i(t)| + |LC_j(t)| \leq C_i + C_j - 1 \qquad (5)$$

Example 2 We elucidate the definition of RO DAP, by applying it on the small system of Figure 2. The ordering used in the policy implementation is the *natural* ordering of the system resources, i.e., $o(R_i) = i$, $\forall i$. Furthermore, we observe that job instances executing the last stage of their route can never deadlock the system, since their unloading from the system is always a feasible step. Hence, they can be ignored during the evaluation of the admissibility of a resource allocation state, and therefore, they are omitted during the definition of the content of $RC_i(t)$ and $LC_i(t)$.[7]

It is easy to see that job stages JT_{11}, JT_{12} as well as JT_{31} are right-directed, while job stage JT_{21} is undirected, and job stage JT_{22} is left-directed. Hence, the contents of the RC_i and LC_i counts are defined by the following table:

Resource	RC_i	LC_i
R_1	JT_{11}, JT_{31}	
R_2	JT_{12}	
R_3	JT_{21}	JT_{21}
R_4		JT_{22}

[7] Although not used in the previous example, a similar remark regarding the (in-)significance of last job stages in deadlock avoidance applies to the implementation of RUN DAP.

Then, part 3 of Definition 3 implies that this implementation of RO on the considered RAS imposes the following set of linear inequalities on the system state:

$$\begin{bmatrix} 1 & & & & 1 \\ 1 & & 1 & & 1 \\ 1 & & & 1 & 1 \\ & 1 & 1 & & \\ & 1 & & 1 & \\ & & 1 & 1 & \end{bmatrix} \mathbf{s}(t) \leq \begin{bmatrix} 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{6}$$

In Equation 6, each inequality corresponds to a pair (R_i, R_j) with $R_i < R_j$, and with all these pairs ordered lexicographically in increasing order (i.e., (R_1, R_2), (R_1, R_3), etc.). □

The policy implementations undertaken in Examples 1 and 2 clearly demonstrate that, both, RUN and RO DAP's can be expressed as a set of linear inequalities on the system state, i.e.,

$$A_P \mathbf{s}(t) \leq f_P(\mathbf{C}), \quad P \in \{RUN, RO\} \tag{7}$$

where A_P is an incidence (0/1) matrix, expressing the policy-defining logic, and $f_P()$ is a vector function of the resource capacities. It is this algebraic characterization of the policy logic that allows the formulation and solution of the job rerouting problem developed in the next sections.

3.2 The "Optimal" Job Rerouting Problem: Statement and Formulation

Problem Statement A verbal description of the problem addressed in this section is as follows: Consider an FMS configuration currently executing a set of job instances $j_j \in \mathcal{I}$. Each job j_j is pre-assigned to a certain route $JR_j \in \mathcal{JR}_j$, and it is currently executing stage $JT_{j,k(j)}$ in that route. Furthermore, suppose that the FMS is structurally controlled by either RUN or RO, and hence, the constraints expressed by the corresponding Equation 7 are observed by the system operation. At some point, one of the following two contingencies occurs: (i) one of the system resources fails, and is not going to be operable for a considerably long time, or (ii) an expedient job arrives, which requires immediate processing through the system. The occurrence of any of these two events might lead to a resource allocation status which is not admissible by the applied policy any more (in the case of resource failures, for instance, it might not be even safe, since jobs routed towards the failing resource will not be able to complete processing). This implies that some jobs should be temporarily unloaded from the system, to bring it back to a policy-admissible (and thus, safe) state. We would like to minimize this disruption by re-routing currently executing jobs.

Analytical formulation We proceed with the analytical formulation of this problem, as follows: Let $x_{jq} = 1$ if job instance $j_j \in \mathcal{I}$ is assigned to its q-th route, $q \in \{1, \ldots, |\mathcal{R}_{j,k(j)} \times \mathcal{R}_{j,k(j)+1} \times \ldots \times \mathcal{R}_{j,l(j)}|\}$; 0, o.w. Furthermore, let $\mathbf{x}$ denote the vector aggregating all variables x_{jq}. Then, we seek to:

$$\max z = \sum_{j,q} x_{jq} \tag{8}$$

s.t.

$$\bar{A}_P \mathbf{x} \leq f_P(\mathbf{C}) \tag{9}$$

$$\sum_q x_{jq} \leq 1, \quad \forall j \tag{10}$$

$$x_{jq} \in \{0, 1\}, \quad \forall j, q \tag{11}$$

The objective function of this formulation states our intention to minimize the number of running jobs unloaded from the system. Constraint 9 requires policy admissibility for the resulting resource allocation state. Notice that the policy-defining matrix in this constraint is technically different from that of Equation 7, since each column now corresponds not to a distinct route stage, but to a job instance running through the system executing a certain route stage. Specifically, for the case of RUN DAP, the column in $\bar{A}_P$ corresponding to variable x_{jq} is a replication of the A_{RUN} column corresponding to route stage JT^q_{jk}, i.e., the k-th stage of job type JT_j when implemented through the q-th possible route. For the case of RO DAP, $\bar{A}_P$ incorporates two different requirements, and it can be considered as the vertical concatenation of two different sub-matrices: The first sub-matrix has one row for every system resource, and expresses the requirement that no resource is over-occupied in the final solution. The second sub-matrix is an expression of the constraints imposed by the RO logic, and it can be synthesized in a way similar to that suggested for the RUN DAP, above.[8] Constraint 10 requires that each job instance is assigned to at most one route. Finally, constraint 11 states the binary nature of our decision variables.

The above formulation covers the problem of the job rerouting in case of machine failures by appropriately synthesizing vector $f_P(\mathbf{C})$ to reflect the new resource availability. The problem of expediting a newly arrived job j^*_j is addressed by including this job in the set of currently

[8] Notice that in the case of RUN DAP, the constraints preventing the over-allocation of the neighborhood capacities establish automatically that no resource is over-allocated in the final solution (since every resource belongs in its neighborhood). This requirement is not inherent, however, in the RO logic, and therefore, it must externally imposed. The detailed structure of the $\bar{A}_P$ and $f_P(\mathrm{C})$ matrices for RUN and RO DAP's is further elucidated in the examples of Section 3.4.

executing jobs $\mathcal{I}$, and setting $\sum_q x_{j^*q} = 1$ in the corresponding constraint 10.[9] Finally, notice that the proposed formulation can accommodate a broader scheme of decision making, according to which the completion of currently running jobs has varying degrees of urgency, by associating a weight w_j with each job $j_j \in \mathcal{I}$, and attempting to maximize the weighted sum:

$$\max z' = \sum_{j,q} w_j x_{jq} \tag{12}$$

This is the objective to be used in the following discussion. The resulting formulation will be referred to as the *"Optimal" Job Rerouting* problem.

3.3 Solving the "Optimal" Job Rerouting Problem

Formulation (12, 9 – 11) is an integer (binary) program (IP). Hence, the increased problem complexity is already determined by the binary nature of the problem decision variables. Typically, the solution of this IP would be addressed through an IP-solving technique, like Branch & Bound [37]. The problem solution is complicated even further by the fact that the length of vector $\mathbf{x}$ – and therefore the number of columns of matrix $\bar{A}_P$ – is $O(K^{\bar{I}} \sum_i C_i)$. In other words, explicit enumeration of all the problem variables during the solution of the relaxing LP's in the Branch & Bound implementation is prohibitive. Therefore, next we propose the application of a *column-generation* scheme [37] for handling this problem.

Let $\bar{I} \subseteq \mathcal{I}$ denote the set of jobs unassigned to job routes at a certain stage of the Branch & Bound enumeration. Then, only jobs $j_j \in \bar{I}$ should be considered in the formulation of the LP relaxation of that stage, since the effect of assigning the jobs in $\mathcal{I} \setminus \bar{I}$ to specific routes can be introduced in the formulation by appropriately adjusting the right-hand-side vector $f_P(\mathbf{C})$. Also, the binding of a specific variable x_{jq} to zero for some job $j_j \in \bar{I}$, implies the elimination of the q-th route of that job from further consideration. For the remaining job-route pairs (j, q), let $\mathcal{CR}$ denote the set of job routes currently introduced explicitly in the LP formulation.[10] The proposed column relaxation scheme will first solve optimally the LP relaxation defined by equations 12, 9, 10 and $x_{jq} \geq 0$ on the restriction of $\bar{A}$ on $\mathcal{CR}$. Then, it searches for other

[9] A similar formulation can address the more general problem of whether *any* new job can be loaded into the system. Specifically, in this case, we set $\sum_q x_j = 1$ for all currently loaded jobs, and we also account for the fact that these jobs are currently loaded in certain resources by adjusting appropriately the right-hand-side vector $f_P(\mathrm{C})$. Hence, our initial assumption that the currently running jobs $j_j \in \mathcal{I}$ have been pre-assigned to specific routes is justified.

[10] For the initial LP-relaxation of the Branch & Bound algorithm, this set might be the set of current job routes which are still feasible. For subsequent nodes, one can use the columns existing in parent nodes to initialize the Simplex tableau.

possible routes that can improve the current solution, by solving, for each job $j_j \in \bar{I}$, the problem:

$$\begin{aligned}\rho_j &= \min_q \{\lambda_{\bar{A}}^T \bar{A}_{[\cdot,jq]} + \lambda_j - w_j\} \\ &= \min_q \{\lambda_{\bar{A}}^T \bar{A}_{[\cdot,jq]}\} + \lambda_j - w_j \qquad (13)\end{aligned}$$

In Equation 13, $\lambda_{\bar{A}}$ is the projection of the *Simplex Multiplier* vector (*shadow prices* [37]), λ, of the current LP solution, on the constraints generated by Equation 9. λ_j is the Simplex Multiplier corresponding to the Constraint 10 generated for job j_j, and w_j is the objective coefficient of job j_j, expressing its urgency. Notice that ρ_j is the *reduced cost* of variable x_{jq} in the Simplex logic, and therefore, if $\rho_j < 0$ for some jobs j_j, the corresponding minimizing columns are introduced into the Simplex tableau, and the LP-relaxation optimization procedure is resumed; o.w., it is concluded that the current pricing of x_{jq}'s is optimal for that node.

It remains to define a procedure for solving the subproblem defined by Equation 13, for the cases of, both, RUN and RO. Notice that this procedure will be policy-dependent, since the detailed solution depends on the structure of the $\bar{A}_{[\cdot,jq]}$ columns, and thus, the policy-defining logic. We examine each case separately.

The column generation scheme for RUN DAP: We compute

$$u = \min_q \{\lambda_{\bar{A}}^T \bar{A}_{[\cdot,jq]}\} \qquad (14)$$

through a *forward dynamic programming (DP)* scheme [37]. In this DP model, the problem *stages* correspond to the remaining route stages $JT_{jk}, JT_{j,k+1}, \ldots, JT_{j,l(j)}$, while the set of *states* associated with stage JT_{jp}, $p = k, \ldots, l(j)$, corresponds to the set of possible options for executing that stage, $\mathcal{R}_{jp}$. In the following, we shall denote the l-th option (resource) in this set by R_{jp}^l. Furthermore, let $V(R_{jp}^l)$ denote the *value* of option R_{jp}^l, i.e., the minimal cost over the set of feasible routes implementing subsequence $< JT_{j,k(j)}, \ldots, JT_{jp} >$, with stage JT_{jp} supported by resource R_{jp}^l. Then, we have the following theorem:

Theorem 1 *The value function of the optimal policy for the forward DP problem defined by Equation 14, is given by the recursion:*

$$V(R_{jp}^l) = \min_x \{V(R_{jp}^l | R_{j,p-1}^x)\} \qquad (15)$$

where:

$$V(R^l_{jp}|R^x_{j,p-1}) = \begin{cases} V(R^x_{j,p-1}), \text{ if the optimal route} \\ \text{to } R^x_{j,p-1} \text{ includes: (i) a resource} \\ \text{with capacity higher than } C_{R^l_{jp}}, \\ \text{or (ii) } R^l_{jp}, \text{ itself;} \\ \min\{V(R^x_{j,p-1}) + \lambda_{R^l_{jp}}, \\ \min_{(t,y):\ C_{R^y_{j,p-t}} > C_{R^l_{jp}}} \{V(R^y_{j,p-t})\} \\ \}, \text{ o.w.} \end{cases} \tag{16}$$

Proof: Function $V(R^l_{jp}|R^x_{j,p-1})$ in Theorem 1 denotes the value of the optimal route that reaches resource R^l_{jp} (i.e., the l-th option of stage JT_{jp}) through resource $R^x_{j,p-1}$ (the x-th option of stage $JT_{j,p-1}$).

The correctness of Equation 16 for computing this function is established on the basis of the following two key observations:

1. The (synthetic) cost $\lambda^T_{\bar{A}} \bar{A}_{[\cdot,jq]}$ of the routing option defined by column $\bar{A}_{[\cdot,jq]}$ expresses the cost of the reservations posed by this route on the system resources, with the unit cost of each resource being defined by the corresponding Simplex multiplier.

2. According to the definition of the resource neighborhoods (c.f., Section 3.1), a job instance j_j is in the neighborhood of resource R_k supporting a subsequent stage, and therefore, reserves a unit of it, only if the route segment between the current supporting resource (inclusive) and resource R_k does not contain a resource R_d with $C_{R_d} > C_{R_k}$.

Then, part (i) of the first branch in Equation 16 follows immediately from the previous remarks, since no additional reservation on resource R^l_{jp} must be made. Part (ii) covers the case that resource R^l_{jp} has already been revisited in the optimal route to $R^x_{j,p-1}$. There are two possibilities:

i. Job j_j is already in the neighborhood of R^l_{jp} because of this first visit. Therefore, $\lambda_{R^l_{jp}}$, the shadow price of the RUN constraint corresponding to resource R^l_{jp}, has already been added to $V(R^x_{j,p-1})$, and $V(R^l_{jp}|R^x_{j,p-1}) = V(R^x_{j,p-1})$.

ii. Job j_j is not already in the neighborhood of R^l_{jp}. This implies – according to the above remark regarding the job inclusion in resource neighborhoods – that the segment of the optimal route to $R^x_{j,p-1}$ preceding the first visit to R^l_{jp} contains a resource R_d with $C_{R_d} > C_{R^l_{jp}}$. But then, the same resource R_d implies that job j_j must not be included in the neighborhood of resource R^l_{jp} because of this second visit. Hence, again $V(R^l_{jp}|R^x_{j,p-1}) = V(R^x_{j,p-1})$.

To prove the validity of the second branch of Equation 16, we notice that its defining condition implies that if resource R^l_{jp} is reached through the optimal route to resource $R^x_{j,p-1}$, job j_j must be introduced in the neighborhood of R^l_{jp}. Therefore, the value of the resulting route is $V(R^x_{j,p-1}) + \lambda_{R^l_{jp}}$. However, if resource $R^x_{j,p-1}$ is reached through a resource $R^y_{j,p-t}$ with capacity $C_{R^y_{j,p-t}} > C_{R^l_{jp}}$, then $V(R^l_{jp}|R^x_{j,p-1}, R^y_{j,p-t}) = V(R^x_{j,p-1}|R^y_{j,p-t}) = V(R^y_{j,p-t})$. The last equality holds because the defining assumption of the second branch in Equation 16 implies that the optimal route to $R^x_{j,p-1}$ contains a route segment from stage $JT_{j,p-t+1}$ to stage $JT_{j,p-1}$, with resources of capacity lower than or equal to $C_{R^l_{jp}}$, and therefore, strictly lower than $C_{R^y_{j,p-t}}$; hence, option $R^x_{j,p-1}$ can be accessed from option $R^y_{j,p-t}$ at no additional cost. Since it is possible that $V(R^x_{j,p-1}) < V(R^y_{j,p-t})$ but $V(R^x_{j,p-1}) + \lambda_{R^l_{jp}} > V(R^y_{j,p-t})$, we must also consider these alternative routes that reach R^l_{jp} through $R^x_{j,p-1}$ *and* $R^y_{j,p-t}$ in the computation of $V(R^l_{jp}|R^x_{j,p-1})$; this necessitates the internal $\min\{\}$ in the second branch of Equation 16. $\square$

An upper bound for the complexity of this DP scheme can be obtained as follows: The evaluation of Equation 16 for a certain pair $(R^l_{jp}, R^x_{j,p-1})$ is of complexity no higher than $O(Kp)$, where K is an upper bound to the number of implmentational options associated with a single processing stage; this complexity arises primarily from the evaluation of $\min_{(t,y):\ C_{R^y_{j,p-t}} > C_{R^l_{jp}}} \{ V(R^y_{j,p-t})\}$, in that formula. The complexity of evaluating $V(R^l_{jp})$, the value of the l-th option for the p-th processing stage of job instance j_j, is $O(K^2p)$, and the complexity of evaluating $V(R^l_{jp})$, $l = 1, \ldots, |\mathcal{R}_{jp}|$, for all options associated with stage p is $O(K^3p)$. But then, the complexity of evaluating all options for all stages for a given job instance j_j is $O(K^3 \sum_{p=1}^{\bar{l}} p) = O(K^3\bar{l}^2)$.

The column generation scheme for RO DAP: An efficient procedure for solving the problem defined by Equation 14 in the RO case, is defined by the following theorem:

Theorem 2 *Given a job instance j_j, executing stage $JT_{j,k(j)}$ with support set $\mathcal{R}_{j,k(j)}$, the column $\bar{A}_{[\cdot,jq]}$ minimizing the right-hand-side of Equation 14, is obtained by the following procedure:*

i. For every resource $R_l \in \mathcal{R}_{j,k(j)}$,

(i) check the existence of a right-directed route for job j_j beginning with resource R_l; in case of a positive answer, compute its cost:

$$V_r(R_l) = \lambda_{R_l} + \sum_{q:\ R_q > R_l} \lambda_{(R_l,R_q)} \tag{17}$$

(ii) check the existence of a left-directed route for job j_j beginning with resource R_l; in case of a positive answer, compute its cost:

$$V_l(R_l) = \lambda_{R_l} + \sum_{q:\ R_q < R_l} \lambda_{(R_q, R_l)} \tag{18}$$

(iii) If no right or left-directed route exists, then job instance j_j supported by resource R_l is an undirected job, and the associated cost is:

$$V_u(R_l) = \lambda_{R_l} + \sum_{q:\ R_q \neq R_l} \lambda_{(R_q, R_l) \vee (R_l, R_q)} \tag{19}$$

(iv) The cost of resource R_l (i.e., the l-th option) is:

$$V(R_l) = \min\{V_r(R_l), V_l(R_l), V_u(R_l)\} \tag{20}$$

where undefined argument costs in the $\min\{\}$ *expression are considered to be equal to* ∞.

ii. Then,

$$u = min_l\{V(R_l)\} \tag{21}$$

Proof: The truth of the above theorem results immediately from the following observation: The first sub-matrix of $\bar{A}_{RO}$ contains a row for each resource R_l, while the second sub-matrix of $\bar{A}_{RO}$ contains a row for every pair of resources (R_k, R_l) with $o(R_k) < o(R_l)$. Furthermore, a right-directed job instance, currently supported by resource R_{k*}, contributes a column in $\bar{A}_{RO}$ with unit entries in the row corresponding to resource R_{k*} in the first sub-matrix, and in all rows corresponding to pairs (R_{k*}, R_l) in the second sub-matrix. Similarly, a left-directed job contributes a column with unit entries in the row of R_{k*} of the first sub-matrix, and all rows corresponding to pairs (R_l, R_{k*}) in the second sub-matrix. Finally, an undirected job supported by resource R_{k*} contributes an $\bar{A}_{RO}$-column with unit entries in the row R_{k*} of the first sub-matrix, and in all rows involving resource R_{k*} either as left or right resource in the second sub-matrix. Hence, every right-directed/left-directed/undirected route for a job currently supported by resource R_{k*} has a cost provided by Equation 17/ 18/ 19, with $R_l \equiv R_{k*}$.
□

It remains to define a procedure for testing the existence of a right or left-directed route for job j_j, when it is supported by resource R_l. The test concerning the existence of a right-directed route can be performed in a constructive manner as follows:

$q = k(j) + 1;$

`repeat` $\{$

$$R_{jq} = \begin{cases} \arg\min_{l\ :\ o(R^l_{jq}) > o(R_{j,q-1})} \{o(R^l_{jq})\}, \\ \textit{if } \{R_l\ :\ o(R^l_{jq}) > o(R_{j,q-1})\} \neq \emptyset; \\ \texttt{null},\ o.w. \end{cases}$$

$q = q + 1;$

$\}$ `until` $(R_{jq} = \texttt{null} \vee q > l(j))$

Exiting the above loop because of the first condition implies a negative answer, while exiting because of the second condition implies a positive one. The procedure for the existence of a left-directed route is developed in a similar fashion. In fact, the two tests can be carried out together, in a single-run, at a computational cost of $O(K\bar{l})$, where K is an upper bound on the implementational options for a single job stage, and $\bar{l}$ is an upper bound on the job route length. On the other hand, the computation implied by Equations 17 to 19 is of complexity $O(|\mathcal{R}|) = O(m)$, since it only involves the pairs of R_l with any other resource. Hence, the overall complexity of evaluating Equation 20, for a single option, l, of job instance j_j, is $O(K\bar{l} + m)$. Then, evaluating Equation 21 is of complexity $O(K(K\bar{l} + m)) = O(K^2\bar{l} + Km)$.

3.4 Examples

We conclude this section with two examples that highlight the IP formulations of the "optimal" job rerouting problem, and the solution procedures proposed herein.

Example 3 The system to be considered in this example is the RAS of Figure 2, controlled by the RUN DAP. Assume that currently this system is loaded with: (i) an instance, j_1, of job type JT_1 executing its first stage, (ii) an instance, j_2, of job type JT_2 executing its first stage, and (iii) an instance, j_3, of job type JT_3 executing its second stage. At this point, resource R_2 fails, and its currently assigned job j_3 is scrapped. Furthermore, it is obvious that the current route of job instance j_1 is infeasible. The situation is depicted in Figure 3.

However, let us suppose that resource R_4 can function as an alternative for job stage JT_{12}, resource R_1 is an alternative for job stage JT_{13}, and resource R_1 is an alternative for stage JT_{22}. Hence, job instance j_1 has four routing options for its remaining processing, i.e., $o_{11}:\ R_2 \rightarrow R_3,\ o_{12}:\ R_2 \rightarrow R_1,\ o_{13}:\ R_4 \rightarrow R_3,\ o_{14}:\ R_4 \rightarrow R_1$, where in the context of this example, o_{ij} denotes the j-th routing option of job instance j_i. The routing options for job instance j_2 are: $o_{21}:\ R_4 \rightarrow R_3,\ o_{22}:\ R_1 \rightarrow R_3$. As a result, the complete IP formulation (c.f., 8– 11) for this

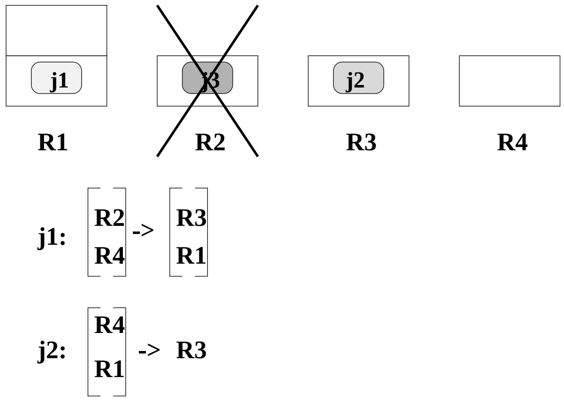

Figure 3: Example: An “Optimal” Rerouting problem

small example is:

$$\max x_{11} + x_{12} + x_{13} + x_{14} + x_{21} + x_{22}$$

s.t.

$$\begin{aligned}
\begin{bmatrix} & 1 & & 1 & & 1 \\ 1 & 1 & & & & \\ 1 & & 1 & & 1 & \\ & & 1 & 1 & 1 & \end{bmatrix}
\begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \end{bmatrix} &\le \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \end{bmatrix} \\
x_{11} + x_{12} + x_{13} + x_{14} &\le 1 \\
x_{21} + x_{22} &\le 1 \\
x_{ij} \in \{0, 1\} \quad \forall i, j & \qquad (22)
\end{aligned}$$

Notice that the above formulation still considers resource R_2, and forces the failing condition of that resource by setting its nominal capacity to zero. A more (computationally) efficient formulation of this particular situation would have been to eliminate resource R_2 completely from our model. We opted to use the formulation of Equation 22 in the development of our example since this approach reflects the more general case of accommodating the failure of one unit from a multi-capacity resource (i.e., the resource capacity is reduced by one, but the resource type does not vanish). Furthermore, we consider the ability of the model to accommodate effectively the rather singular case of zero-capacity resources as a good indication of its robustness. We want also to emphasize that the formulation of Equation 22 assumes that all parts in the system have completed the processing of their current stage, and therefore, they are ready to be advanced to the resource that will support their next processing step. In case that there exist parts in the system that require significant time to the completion of their current stage, their non-relocatability can be introduced into the model by (i) requiring that these parts remain in the system (i.e., the corresponding constraint in Equation 10 should be turned into equality), and (ii) "forcing" all the routes associated with these parts to start from the currently supporting resource, R_q (i.e., R_q should be the only option for the first stage of each alternative route).

It is easy to see, by inspection, that the optimal solution (route assignment) for the model of Equation 22, is $o_{13}: \; R_4 \rightarrow R_3$ or $o_{14}: \; R_4 \rightarrow R_1$ for j_1, and $o_{22}: \; R_1 \rightarrow R_3$ for j_2. This result was also verified through the LINDO IP solver. To apply the column generation scheme proposed in Section 3.3, we start with the IP corresponding to the current job routes, o_{11} and o_{21}. Hence, the initial LP relaxation is:

$$\max x_{11} + x_{21}$$

s.t.

$$\begin{bmatrix} & \\ 1 & \\ 1 & 1 \\ & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \end{bmatrix} \leq \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$x_{11} \leq 1$$

$$x_{21} \leq 1$$

$$0 \leq x_{ij} \leq 1 \quad \forall i, j \tag{23}$$

The optimal solution to this LP is: $x_{11}^1 = 0$, $x_{21}^1 = 1$, and the optimal shadow prices are: $\lambda^1 = [0\ 0\ 1\ 0\ 0\ 0]$. The column-generating DP for job j_1 has two stages with two states each, corresponding to the system resources as follows:[11]

State	Stage 1	Stage 2
1	R_2	R_3
2	R_4	R_1

[11] In this discussion, superscripts indicate the running iteration in the proposed column generation scheme, while subscripts indicate the processed job instance. As an example, ρ_j^i denotes the minimum reduced cost over the routes of job instance j_j, during the i-th column generating iteration. In particular, the notation $V_j^i(p, q)$ denotes the optimal value function of the q-th state in the p-th stage, for the DP problem formulated for job instance j_j, during the i-th column generating iteration.

The optimal value function for the above shadow prices, λ^1, is: $V_1^1(1,1) = V_1^1(1,2) = 0$, $V_1^1(2,1) = \min\{V_1^1(2,1|1,1) + \lambda_{R_3}, V_1^1(2,1|1,2) + \lambda_{R_3}^1\} = 1$, $V_1^1(2,2) = \min\{V_1^1(2,2|1,1) + \lambda_{R_1}^1, V_1^1(2,2|1,2) + \lambda_{R_1}^1\} = 0$. Hence, $u_1^1 = 0$, corresponding to routes $o_{12} : R_2 \rightarrow R_1$, $o_{14} : R_4 \rightarrow R_1$. The associated reduced cost is $\rho_1^1 = u_1^1 + \lambda_1^1 - w_1 = 0 + 0 - 1 = -1$, and therefore, variables x_{12} and x_{14} enter the LP-relaxation tableau.

Similarly, for job j_2, the underlying DP structure is given by the table:

	Stage	
State	1	2
1	R_4	R_3
2	R_1	–

and the optimal value function is: $V_2^1(1,1) = V_2^1(1,2) = 0$, $V_2^1(2,1) = \min\{V_2^1(2,1|1,1) + \lambda_{R_3}^1, V_2^1(2,1|1,2)\} = \min\{1,0\} = 0$. Hence, $u_2^1 = 0$, corresponding to route $o_{22} : \ R_1 \rightarrow R_3$. The associated reduced cost $\rho_2^1 = u_2^1 + \lambda_2^1 - w_2 = 0 + 0 - 1 = -1$, and therefore, variable x_{22} also enters the LP-relaxation tableau. Hence, the extended LP is:

$$\max x_{11} + x_{21} + x_{12} + x_{14} + x_{22}$$

s.t.

$$\begin{bmatrix} & & 1 & 1 & 1 \\ 1 & & 1 & & \\ 1 & 1 & & & \\ & 1 & & 1 & \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{12} \\ x_{14} \\ x_{22} \end{bmatrix} \leq \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$x_{11} + x_{12} + x_{14} \leq 1$$

$$x_{21} + x_{22} \leq 1$$

$$0 \leq x_{ij} \leq 1 \quad \forall i,j \tag{24}$$

The optimal solution for this LP is $x_{14}^2 = x_{22}^2 = 1$, $x_{11}^2 = x_{12}^2 = x_{21}^2 = 0$, and $\lambda^2 = [0\ 0\ 0\ 0\ 1\ 1]$. Therefore, the new optimal value functions for the underlying DP's are uniformly equal to 0, and $\rho_1^2 = u_1^2 + \lambda_1^2 - w_1 = 0 + 1 - 1 = 0$, $\rho_2^2 = u_2^2 + \lambda_2^2 - w_2 = 0 + 1 - 1 = 0$. Therefore, the current LP-relaxation tableau is optimal, and since the values of all the x_{ij} variables are integral, we have an optimal routing, i.e., j_1 is routed through $R_4 \rightarrow R_1$ and j_2 through $R_1 \rightarrow R_3$. The optimal rerouting pattern is depicted in Figure 4.

Notice that under the second solution approach, route $o_{13} : \ R_4 \rightarrow R_3$ was never explicitly considered in the IP formulations. It is this kind of computational economies that we expect to achieve through the column generation scheme proposed in this section. □

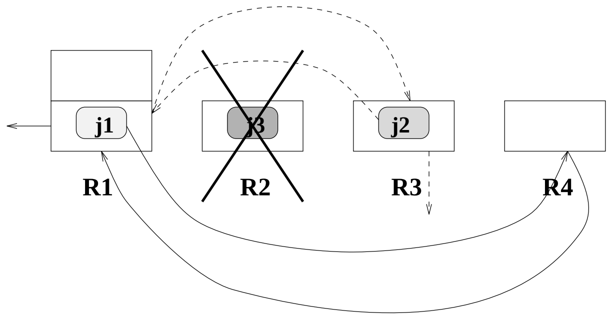

Figure 4: Example: An "Optimal" Rerouting Scheme

Example 4 This example demonstrates the accommodation of expedient ("hot") jobs by means of the proposed algorithms, and it delineates the formulation and solution of the "Optimal Job Rerouting" problem for the case that the applied structural control policy is RO DAP. As before, consider the system of Figure 2, supporting three job types, JT_1, JT_2 and JT_3, with the following flexible routings:

$$\begin{aligned} JT_1 &: R_1 \rightarrow \left\{ \begin{array}{c} R_2 \\ R_4 \end{array} \right\} \rightarrow \left\{ \begin{array}{c} R_3 \\ R_1 \end{array} \right\} \\ JT_2 &: R_3 \rightarrow \left\{ \begin{array}{c} R_4 \\ R_1 \end{array} \right\} \rightarrow R_3 \\ JT_3 &: R_1 \rightarrow R_2 \end{aligned} \tag{25}$$

Currently, there are three jobs running through the system: j_1, j_2 and j_3. Job j_1 has just finished processing stage JT_{11} on resource R_1, and job j_2 has finished processing stage JT_{21} on resource R_3. Job j_3 is executing stage JT_{32} on resource R_2. Clearly, the current state is admitted by the RO DAP, since job j_3 executing its last stage is not actually considered in the policy-imposed job countings, and as a result, from the policy perspective, only resource R_3 is allocated to capacity. Consider, however, that at this point, a "hot" job, j_4, of type JT_1, arrives. This job requests immediate loading to resource R_1, but it is easy to see that such an action is prohibited by RO DAP, since it violates the policy constraint for the resource pair (R_1, R_3). Hence, we are faced with an optimal job rerouting problem.

The routing options of the four jobs can be easily derived from the routing information of Equation 25, and their current processing status; for this a small example, they are enumerated in Table 1. Table 1 provides also a characterization of each routing option as right/left/undirected, based on the natural ordering of the system resources, i.e., $\mathtt{ORDER}(R_i) = i,\ i = 1, \ldots, 4$. Furthermore, it has been estimated that finishing job j_2 is more urgent than

j_1			j_2			j_3			j_4		
o_{11}:	$R_2 \to R_3$	R	o_{21}:	$R_4 \to R_3$	L	o_{31}:	R_2	-	o_{41}:	$R_1 \to R_2 \to R_3$	R
o_{12}:	$R_2 \to R_1$	L	o_{22}:	$R_1 \to R_3$	R				o_{42}:	$R_1 \to R_2 \to R_1$	U
o_{13}:	$R_4 \to R_3$	L							o_{43}:	$R_1 \to R_4 \to R_3$	U
o_{14}:	$R_4 \to R_1$	L							o_{44}:	$R_1 \to R_4 \to R_1$	U

Table 1: Example 4: The job routing options

finishing job j_1. Finally, since job j_3 is in its last processing stage, it should be interrupted only if it is absolutely necessary.

The above description of the given situation leads to the following algebraic formulation:

$$\max 1(x_{11} + x_{12} + x_{13} + x_{14}) + 2(x_{21} + x_{22}) + 10x_{31} + 1(x_{41} + x_{42} + x_{43} + x_{44}) \tag{26}$$

s.t.

$$\begin{bmatrix} & & & & & 1 & & 1 & 1 & 1 & 1 \\ 1 & 1 & & & & & 1 & & & & \\ & & & & & & & & & & \\ & & 1 & 1 & 1 & & & & & & \\ & 1 & & & & 1 & & 1 & 1 & 1 & 1 \\ & & & & & 1 & & 1 & 1 & 1 & 1 \\ & & 1 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 \\ 1 & & & & & & & & & & \\ 1 & & 1 & 1 & 1 & & & & & & \\ & & 1 & 1 & 1 & & & & & & \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{31} \\ x_{41} \\ x_{42} \\ x_{43} \\ x_{44} \end{bmatrix} \leq \begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{27}$$

$$\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} &\leq 1 \\ x_{21} + x_{22} &\leq 1 \\ x_{31} &\leq 1 \\ x_{41} + x_{42} + x_{43} + x_{44} &= 1 \end{aligned} \tag{28}$$

$$x_{ij} \in \{0,\ 1\}, \quad \forall i, j \tag{29}$$

Regarding the formulation of Equations 26 – 29, we make the following remarks:

1. Notice that the aforestated preferences regarding the unloading of the currently running jobs are introduced in the formulation by setting higher criticality weights for the jobs that we would like to remain in the system.

2. On the other hand, the requirement that the "hot" job j_4 is expedited through the system is expressed by setting the corresponding constraint 28 to an equality constraint; hence job j_4 will be definitely loaded in the system.

3. As it was indicated in Section 3.2, the first four rows of Constraint 27 correspond to resources R_1 to R_4, expressing the requirement that none of these resources is over-allocated in the final solution. The remaining six rows correspond to resource pairs (R_1, R_2) to (R_3, R_4), enumerated lexicographically, and express the RO logic, as applied to the job routing options of Table 1. Also, for this particular formulation, the constraint expressed by row 3 (over-allocation of resource R_3) is trivially satisfied since no routing option is using this resource, and therefore, it will be dropped in the subsequent discussion.

4. Closer observation of the structure of Equations 26 – 29 reveals that *for any given job instance, all routing options starting with the same resource and leading to same route type regarding the right/left/undirected characterization, correspond to* identical *columns in the "Optimal Job Rerouting" problem, formulated under the RO DAP* (e.g., compare the columns corresponding to routing options o_{13} and o_{14}, which are both left routes starting with resource R_4, and the columns corresponding to routing options o_{42}–o_{44}, which are undirected routes starting with resource R_1). This observation further suggests that for the "Optimal Job Rerouting" problem formulated under the RO DAP, we can "collapse" all these routing options to a single column. This has the additional implication that, given a job instance with K routing options for its next executed stage, only up to $3K$ variables will be introduced in the "Optimal Rerouting" formulation with respect to that job: each of these variables expresses the (potential) right, left and undirected routing options for each starting resource. In fact, considering that right or left routing options, if they are available, will always dominate the undirected option, the number of variables associated with that job instance can be reduced to $2K$. The ultimate implication is that, by exploiting the routing equivalence indicated above, the initial exponential space complexity of the problem formulation can be reduced to *polynomial*: specifically, for a problem instance with $\bar{C}$ job instances involved, each possessing K routing options for the next required processing stage, an upper bound to the number of variables required for the problem formulation is $2K\bar{C}$. Finally, notice that the column generation scheme suggested by Theorem 2 is also applicable to this new formulation; however, given the reduced (polynomial) complexity of this new problem, the computational value of the proposed column generation technique is decreased.

Using the LINDO LP solver, an optimal solution of formulation 26– 29 was found to be: $x_{22} = x_{31} = x_{41} = 1$, and all other variables equal to zero. Hence, the obtained solution requires the unloading of the least urgent job j_1. Indeed, it is easy to verify that j_1 cannot be accommodated into the system: Given that the routing requirements of jobs j_3 and j_4 are practically inflexible with respect to the constraints of formulation 26 – 29, we can see that job j_1 can only be loaded at resource R_4, as a left-directed job. But then, the higher-valued job j_2 can be loaded only to resource R_1 as a right-directed job, and this leads to a violation of RO DAP for the resource pair R_1 and R_4.

To demonstrate the application of the column generation scheme for the "Optimal Rerouting" problem under the RO DAP, suggested by Theorem 2, consider that we start the solution of the above problem focusing initially only to routing options o_{11}, o_{21}, o_{31} and o_{41}. Then, the initial IP to be solved is as follows:

$$\max x_{11} + 2x_{21} + 10x_{31} + x_{41}$$

s.t.

$$\begin{bmatrix} & & & 1 \\ 1 & & 1 & \\ & 1 & & \\ & & & 1 \\ & & & 1 \\ & 1 & & 1 \\ 1 & & & \\ 1 & 1 & & \\ & 1 & & \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{21} \\ x_{31} \\ x_{41} \end{bmatrix} \leq \begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{aligned} x_{11} &\leq 1 \\ x_{21} &\leq 1 \\ x_{31} &\leq 1 \\ x_{41} &= 1 \end{aligned}$$

$$x_{ij} \in \{0,\ 1\}, \quad \forall i, j \tag{30}$$

Notice that in the above formulation, we have removed the vacuous constraint regarding the over-allocation of resource R_3. Also, notice that many constraints in this formulation result in the same analytical expression (inequality), but we must maintained them as separate, in order to facilitate the pricing of additional routing options, not currently introduced in the tableau, that might differentiate between them.

Using LINDO, an optimal solution to the LP relaxation of this problem is found as:[12] $x_{21}^1 = x_{31}^1 = x_{41}^1 = 1$ and $x_{11}^1 = 0$. The shadow price vector corresponding to this solution, is: $\lambda^1 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1 0\ 1]$. Then, applying the logic of Theorem 2 on the routing options of job j_1 (c.f. Table 1), we get: $V_1^1(R_2) = \min\{V_r(R_2),\ V_l(R_2)\} = \min\{1,\ 0\} = 0$, and $V_1^1(R_4) = \min\{V_l(R_4)\} = 1$. Hence, $u_1^1 = \min\{V_1^1(R_2),\ V_1^1(R_4)\} = 0$, and $\rho_1^1 = u_1^1 + \lambda_1^1 - w_1 = 0 + 0 - 1 = -1 < 0$. Therefore, the column corresponding to option o_{12}, that minimizes u_1^1, must be introduced in the tableau. Similarly, for job j_2, we have: $V_2^1(R_4) = \min\{V_l(R_4)\} = 1$, $V_2^1(R_1) = \min\{V_r(R_1)\} = 0$, $u_2^1 = \min\{V_2^1(R_4),\ V_2^1(R_1)\} = 0$, and $\rho_2^1 = u_2^1 + \lambda_2^1 - w_2 = 0 + 1 - 2 = -1 < 0$. Hence, routing option o_{22} must be introduced in the problem formulation. Job j_3 can be omitted since it does not possess any routing alternatives. Finally, for job j_4 it is easy to see that: $V_4^1(R_1) = \min\{V_r(R_1),\ V_u(R_1)\} = 0 = u_4^1$, and therefore, $\rho_4^1 = u_4^1 + \lambda_4^1 - w_4 = 0 + 1 - 1 = 0$. Hence, no new routing options need to be introduced in the tableau.

The new IP formulation to be solved during the second iteration is:

$$\max x_{11} + x_{12} + 2x_{21} + 2x_{22} + 10x_{31} + x_{41}$$

s.t.

$$\begin{bmatrix} & & & 1 & & 1 \\ 1 & 1 & & & 1 & \\ & & 1 & & & \\ & 1 & & 1 & & 1 \\ & & & 1 & & 1 \\ & & 1 & 1 & & 1 \\ 1 & & & & & \\ 1 & & 1 & & & \\ & & 1 & & & \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \\ x_{31} \\ x_{41} \end{bmatrix} \leq \begin{bmatrix} 2 \\ 1 \\ 1 \\ 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{aligned} x_{11} + x_{12} &\leq 1 \\ x_{21} + x_{22} &\leq 1 \\ x_{31} &\leq 1 \\ x_{41} &= 1 \end{aligned}$$

[12] Similar to the previous example, superscripts, i, count column generating iterations, while subscripts, j, refer to the various job instances.

$$x_{ij} \in \{0,\ 1\}, \quad \forall i, j \tag{31}$$

For the sake of brevity, we omit the discussion of the following iterations. It is interesting to notice, however, that the algorithm has already identified an optimal solution during the first iteration, and the subsequent computations requesting the introduction of the new columns in the tableau, essentially results from the *degeneracy* of this optimal solution. □

4 Optimal Job Rerouting for Search-based Ω DAP's

In this section we consider the integration of deadlock avoidance control with routing flexibility, in the case where the DAP-defining condition is evaluated for a given state through polynomial-complexity search on the remaining route segments of running job instances. The main finding is that the search-oriented schemes of these policies do not require the explicit assignment of various job instances to certain routes, and hence, they seem to be more amenable to dynamic job routing than their algebraic counterparts. Under this operational context, the contingency accommodation problem turns into a detection and recovery problem, where the primary issue is finding the "optimal" set of jobs to be unloaded, any time that the occurrence of a certain contingency renders the current resource allocation state non-admissible. We develop these results based on the logic of Banker's DAP [19], while at the end of the section we briefly describe how they apply to the Optimal DAP implementations [17].

4.1 Banker's Algorithm and Routing Flexibility

The classical Banker's algorithm [38] is based on the observation that a state is safe if its running processes can be ordered in such a manner that each process in the ordering can terminate using only its currently allocated resources, resources currently available in the system, and also, resources currently allocated to processes which are preceding it in the order. In the context of correct and scalable FMS DAP's, this idea leads to the concept of the *ordered state*, defined as follows:

Definition 4 *[19] Let $D = \sum_{j=1}^{n} l(j)$. RAS state $\mathbf{s}(t)$ is* ordered *iff there exists an ordering of the set of distinct job route stages, $o() : \{JT_{jk} :\ j = 1, \ldots, n,\ k = 1, \ldots, l(j)\} \rightarrow \{1, \ldots, D\}$, such that the resource requirements for processing to completion a job instance j_i in stage $JT^{(i)}$ can be satisfied by means of the free resources in state $\mathbf{s}(t)$, plus the resources held by job instances j_q in stages $JT^{(q)}$, with $q \leq i$.*

Let V_o denote the set of ordered RAS states. In [19], it is shown that V_o is a strongly connected subspace of the RAS STD, containing the empty state, and therefore, the restriction of the system operation on this set defines a correct DAP.

Definition 4 provides also immediately an algorithm for testing whether a state is ordered or not:

Definition 5 – FMS Banker's Algorithm *[19]*

i. `Set` $\mathcal{UJ} := \{JT_{jk},\ j=1,\ldots,n,\ k=1,\ldots,l(j)\}$; $i=0$; *ORDERED:=TRUE.*

ii. `Repeat`

   (a) $i := i+1$;

   (b) *Try to find a job stage* $JT_{jk} \in \mathcal{UJ}$, *the instances of which can terminate by using their currently allocated resources, plus the currently free resource units.*

   (c) `If` *no such a job stage can be found, ORDERED:= FALSE.*

   (d) `else` $JT_{jk} \equiv JT^{(i)}$; $\mathcal{UJ} := \mathcal{UJ} \backslash \{JT_{jk}\}$; *release the resources held by the job instances of* JT_{jk} *to the pool of the free resource units.*

   `until` $(\mathcal{UJ} = \emptyset) \vee$ *(ORDERED=FALSE)*

iii. `If` *(ORDERED=FALSE)* `return` *FALSE* `else return` *TRUE*

The re-usability of the system resources implies a monotonic increase of the pool of free resources whenever job stages terminate, which makes backtracking unnecessary during the search for a feasible job stage ordering. Hence, the complexity of the above algorithm is polynomial, specifically, $O(mD \log D)$.

Integrating Routing Flexibility to Banker's Algorithm In this paragraph, we show that the logic of Banker's Algorithm presented in Definition 5 can be easily modified to account for flexible job routings, as defined in Section 2. We state this result in the following theorem:

Theorem 3 *Banker's algorithm can be extended to disjunctive Single-Unit RAS where every processing stage for every job is implemented independently from the other stages, and the complexity of the resulting algorithm is not higher than* $O(\bar{l}K \min\{DK, m\bar{C}\})$.

Proof: Since, under flexible routing, every job stage, JT_{jk}, has a number of possible instantiations equal to $\mathcal{R}_{jk}$, the set of entities to be ordered by this modified implementation of Banker's is the set of all *active* instantiations related to all route stages $JT_{jk},\ j=1,\ldots,n,\ k=1,\ldots,l(j)$. Specifically, given a job stage JT_{jk}, we are seeking for every job instance j_j executing this stage, a route in the set $\mathcal{R}_{j,k+1} \times \mathcal{R}_{j,k+2} \times \ldots \times \mathcal{R}_{j,l(j)}$ which can be supported by the currently free resource units, plus the resource $R(j_j) \in \mathcal{R}_{jk}$ which currently supports job j_j. Due to the independence of the implementation of the different processing stages, this test is equivalent to seeking a supporting resource with free units, for every stage $JT_{jq},\ q = k+1, \ldots, l(j)$, for which $R(j_j) \notin \mathcal{R}_{jq}$. The complexity of this test (for a single job instance) is $O(\bar{l}K)$. Furthermore, the number of distinct route stage instantiations in the set of job instances of any given RAS state cannot be higher than $O(\min\{DK, m\bar{C}\})$; the first of these bounds is established by the number of possible distinct instantiations over all route stages, while the second is established by considering the maximum number of jobs that can be in the system at any time. Hence, an upper bound to the complexity of the implementation of Banker's algorithm for Single-Unit RAS with flexible job routing, outlined above, is $O(\bar{l}K \min\{DK, m\bar{C}\})$. □

Notice that the implementation of Banker's algorithm for RAS with dynamic routing, proposed in the proof of Theorem 3, does not require the assignment of running jobs to specific routes. Upon completion of its current processing stage, JT_{jk}, a job j_j can implement its next stage, $JT_{j,k+1}$, on any of the supporting resources $R_q \in \mathcal{R}_{j,k+1}$, as long as the resulting RAS state is ordered.

4.2 Accommodating Operational Contingencies under Banker's algorithm

The non-commitment of running jobs to specific routes under Banker's DAP renders easier the design of reactive policies that accommodate the various contingencies occurring during the system operation. Specifically, all that must be decided upon the occurrence of such a contingency is whether the current resource allocation defines an ordered state, in the new system configuration. A positive answer implies that all jobs can run to completion, in spite of the system reconfiguration, while a negative answer implies that, in order to ensure subsequent smooth operation, some jobs must be (temporarily) unloaded. In the latter case, a naturally arising problem is the identification of a *minimal* set of jobs to be unloaded, where minimality is defined in terms of the weight factors w_i introduced in Equation 12. Mathematically, the problem is defined as follows:

$$\min_{\mathcal{U} \in 2^{\mathcal{I}}} \sum_{j_j \in \mathcal{U}} w_j \tag{32}$$

s.t.

$$\mathbf{s}(\mathcal{I} \backslash \mathcal{U}) \in V_o \tag{33}$$

In this formulation, $\mathcal{I}$ denotes the set of job executed in state $\mathbf{s}$, and $\mathcal{U}$ denotes the set of jobs to be unloaded. Finally, $\mathbf{s}(\mathcal{I} \backslash \mathcal{U})$ denotes the RAS state resulting from $\mathbf{s}$ after eliminating jobs in $\mathcal{U}$. We shall refer to the problem defined by Equations 32, 33 as the *"Optimal" Recovery* problem.

Solving the "Optimal" Recovery problem Define $c() : 2^{\mathcal{I}} \rightarrow \Re^+$ with $c(\mathcal{U}) = \sum_{j_j \in \mathcal{U}} w_j$. The following algorithm solves the "optimal" recovery problem by systematically enumerating the subsets of $\mathcal{I}$ in increasing order w.r.t. $c()$, until a subset $\mathcal{U}^*$ is found s.t. $\mathbf{s}(\mathcal{I} \backslash \mathcal{U}) \in V_o$.

Definition 6 – "Optimal" Recovery Algorithm

i. *Impose a total ordering on the set of jobs, $\mathcal{I}$, which is nondecreasing w.r.t. the job weights, w_j, i.e., define $h() : \mathcal{I} \rightarrow \{1, \ldots, |\mathcal{I}|\}$, s.t. $h(j_i) < h(j_j) \Longleftrightarrow w_{j_i} \leq w_{j_j}$.*

ii. `Set` $\mathcal{A} := \{\{j_j\} : \forall j_j \in \mathcal{I}\}$; *DONE := FALSE.*

iii. `While` *(*`not` *DONE)* `do`

(a) $\mathcal{U}^* := \arg\min_{\mathcal{U} \in \mathcal{A}} \{c(\mathcal{U})\}$

(b) `If` *(*$\mathbf{s}(\mathcal{I} \backslash \mathcal{U}^*) \in V_o$*), DONE:= TRUE.*

(c) `else` $\mathcal{A} := (\mathcal{A} \cup \{\mathcal{U}^* \cup \{j_j\} : \ \forall j_j \in \mathcal{I} \textit{ with } h(j_j) > \max_{j_k \in \mathcal{U}^*}\{h(j_k)\}\}) \setminus \{\mathcal{U}^*\}$

iv. `return` $\mathcal{U}^*$

The following theorem establishes the algorithm correctness.

Theorem 4 *The job set $\mathcal{U}^*$ returned by the "Optimal" Recovery algorithm of Definition 6 is an optimal solution to the "optimal" recovery problem, defined by Equations 32, 33.*

Proof: The validity of the above statement follows immediately from the following lemma:

Lemma 1 *At every iteration of the "optimal" recovery algorithm, $\mathcal{U}^* = \arg\min\{c(\mathcal{U}) : \ \mathcal{U} \in 2^{\mathcal{I}}$, and it has not yet been considered by step (3b) of the algorithm$\}$.*

We prove this lemma by induction on the number of the algorithm iterations. Obviously, it holds true for the first iteration of algorithm, since originally, $c(\mathcal{U})$ is minimized by one of the singletons of $2^{\mathcal{I}}$. Consider that Lemma 1 holds true for all iterations of the algorithm up to iteration k. Hence, $\mathcal{U}_k^* = \arg\min\{c(\mathcal{U}) : \ \mathcal{U} \in 2^{\mathcal{I}} \backslash \{\mathcal{U}_i^*, \ i = 1, \ldots, k-1\}\}$. Then, assuming that at iteration k the condition of step (3b) is false, and the algorithm enters iteration $k+1$, the new minimizing argument of $c()$, $\mathcal{U}_{k+1}^*$, will be either one of the sets in $\mathcal{A}_k \backslash \{\mathcal{U}_k^*\}$, or $\mathcal{U}_k^* \cup \{j_j* = \arg\min\{h(j_j) : \ \forall j_j \in \mathcal{I} \text{ with } h(j_j) > \max_{j_k \in \mathcal{U}_k^*}\{h(j_k)\}\}\}$ (since (i) every other set $\mathcal{U} \notin \mathcal{A}_k$ is a superset of some set in $\mathcal{A}_k$, and (ii) $h()$ is nondecreasing w.r.t. the job weights, w_j). But step (3c) of the algorithm implies that $\mathcal{U}_k^* \cup \{j_j*\} \in \mathcal{A}_{k+1}$. $\square$

The following example provides a more concrete implementation of the algorithm logic.

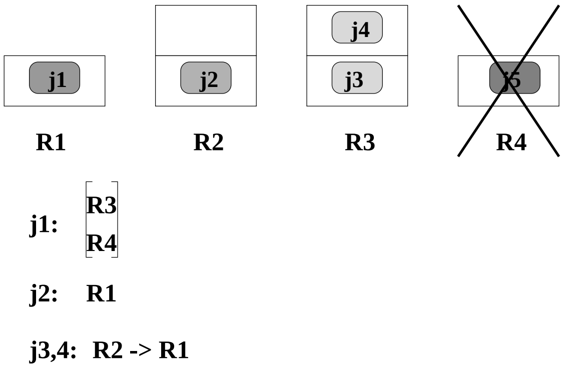

Figure 5: Example: An "Optimal" Recovery problem

Job Instance	running stage	w_j
j_1	JT_{11}	10
j_2	JT_{22}	1
j_3	JT_{21}	5
j_4	JT_{21}	8
j_5	JT_{12}	-

Table 2: Example 5: The RAS state

Example 5 Consider the Single-Unit RAS of Figure 5. This system consists of four resources, R_1, R_2, R_3, R_4, with $C_1 = C_4 = 1$ and $C_2 = C_3 = 2$. In its current configuration, the system supports two job types: $JT_1 =< \{R_1\}, \{R_3, R_4\} >$ and $JT_2 =< \{R_3\}, \{R_2\}, \{R_1\} >$. The current loading of the system is described in Table 2. It is easy to see that the current RAS state is ordered, with a feasible job ordering being $< j_5, j_1, j_2, j_3, j_4 >$. At this point, resource R_4 breaks down, and job j_5 is scrapped. The RAS state of the remaining subsystem $\{R_1, R_2, R_3\}$ is not ordered, and therefore, to bring the system back in the subspace of ordered states, we must unload some of the running jobs, $j_1, \ldots, j_4$.[13]

Application of the "optimal" recovery algorithm on this state evolves as depicted in Figure 6. Job instances are ordered based on their associated weights, i.e., $h(j_1) = 4,\ h(j_2) = 1,\ h(j_3) = 2,\ h(j_4) = 3$. Set $\mathcal{A}$ is initialized to $\mathcal{A}_1 := \{\{j_i\},\ i = 1, \ldots, 4\}$. During the first iteration,

[13] It is worth noticing that this state is actually *safe*, since by advancing j_3 to the idle unit of R_2, job j_1 can proceed to completion, which further allows the other jobs to finish. However, Banker's logic does not accept this type of partial advancements in its search for a feasible completion sequence. This is the price paid to keep the algorithm complexity polynomial in the system size.

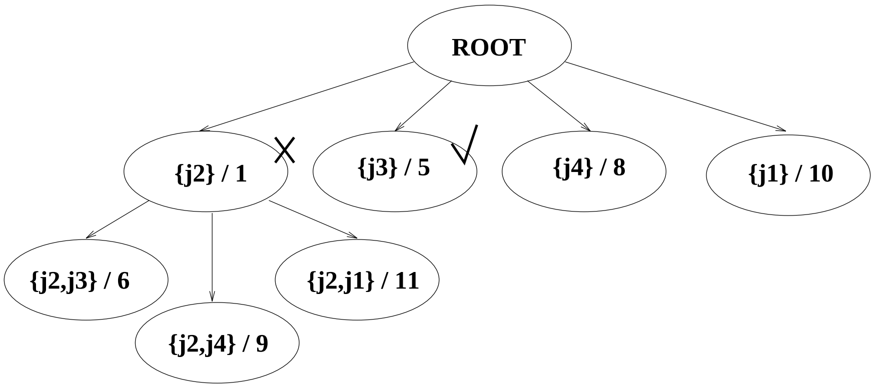

Figure 6: Example: The search-tree developed by the "Optimal" Recovery algorithm

set $\mathcal{U}_1^* = \{j_2\}$, but it can be easily seen that the state resulting by unloading job j_2 is still unordered. Therefore, $\mathcal{A}$ is updated to $\mathcal{A}_2 := \{\{j_i\},\ i = 1,3,4\} \cup \{\{j_2, j_k\},\ k = 1,3,4\}$, and $\mathcal{U}_2^* = \{j_3\}$, with $c(\mathcal{U}_2^*) = 5$. The state resulting by unloading job j_3 is ordered, with a feasible ordering being $< j_1, j_2, j_4 >$. Hence, the optimal set of jobs to be unloaded to get an ordered RAS state is $\mathcal{U}_2^* = \{j_3\}$. $\square$

Improving the Computational Efficiency of the "Optimal" Recovery Algorithm From the algorithm description in Definition 6, it can be seen that, to the extent that it constitutes an enumeration of the power-set of the set of running job instances, $\mathcal{I}$, its worst-case complexity is *exponential*. Here, we consider a number of ways in which we can reduce the cardinality of $\mathcal{I}$ – and therefore, the search space of the algorithm – without compromising the optimality of the obtained solution. The same series of observations make us believe that, in general, the *"critical"* sets of jobs to be unloaded in order to bring the RAS state back in the ordered region will present small cardinality, and therefore, the algorithm will be able to identify the optimal solution by considering only a small subset of $2^{\mathcal{I}}$.

1. The first key observation concerns jobs currently supported by resources with at least one unit of free capacity. Specifically, we have the following proposition:

Proposition 1 *In Single-Unit RAS, removing from the set $\mathcal{I}$ a job j_j, supported by a resource R_i with at least one unit of free capacity, while addressing the "optimal" recovery problem through the algorithm of Definition 6, does not affect the optimality of the obtained solution.*

Proof: The validity of this proposition comes from the observation that, in Single-Unit RAS, any job attempting to proceed to completion through resource R_i, does not require more than one unit of capacity from this resource. Therefore, removing job j_j from the RAS, does not affect the feasibility of these advancements w.r.t. the remaining job instances. □.

As a more concrete example for the applicability of Proposition 1, consider job j_2 in example 5. Since the current existence of a free unit of capacity in resource R_2 does not lead to an ordered state, it is obvious that increasing this capacity to 2 by removing job j_2 from the RAS will not have any improving effect. Notice, however, that eliminating job j_2 from set $\mathcal{I}$ would have spared us all the computational effort related to the deployment of the leftmost subtree emanating from the root node in Figure 6.

2. The second observation concerns jobs which are at the same stage of their routes, supported by the same resource type. Specifically,

Proposition 2 *Suppose that in the considered RAS state there are two job instances j_1, j_2, both of which are currently supported by a fully allocated resource R_i, and they are executing the same job stage JT_{jk}. Furthermore, let $w_1 < w_2$. Then, job j_2 can be eliminated from set $\mathcal{I}$, without compromising optimality.*

Proof: The validity of this claim is a consequence of the fact that, in Single-Unit RAS, to obtain an ordered state, at most one job has to be unloaded from a fully allocated resource R_i (cf. observation 1, above). Thus, containment of j_1 and j_2 in the optimal set $\mathcal{U}^*$ is mutually exclusive. However, for any sets $\mathcal{U}_1$ and $\mathcal{U}_2$ which contain the same set of jobs but for $j_1 \in \mathcal{U}_1$ and $j_2 \in \mathcal{U}_2$, $c(\mathcal{U}_1) < c(\mathcal{U}_2)$, and the claim is established. □

As an example, in the RAS of Figure 5, jobs j_3 and j_4 are both executing stage JT_{21}, supported by resource R_3. Since $w_3 < w_4$, only job j_3 must be included in $\mathcal{I}$. Hence, another node is eliminated from the search tree of Figure 6.

3. One last issue is the treatment of a job j_j which in the current RAS configuration has no feasible route to completion (due to the breakdown of a resource R_i, which is critical to the processing of this job). In such a case, we propose two possible alternatives, depending on the magnitude of the Mean-Time-to-Repair ($MTTR$) of resource R_i, relative to the cumulative (nominal) processing time, $\bar{\tau}$, of the route segment of job j_j between its current stage and the (first) stage that requires resource R_i. If $MTTR \gg \bar{\tau}$, then it might be more reasonable to unload job j_j from the system, before addressing the problem of "Optimal" Recovery through the algorithm of Definition 6. This leads also to another reduction of the set $\mathcal{I}$. On the other hand, if $MTTR \approx \bar{\tau}$, then, there are good chances that resource R_i might be repaired by the time that job j_j requests it, and therefore, unloading immediately this job from the system cannot be justified. However, it might still be reasonable to stipulate the feasibility of advancement of job j_j up to the stage preceding the first visit to resource R_i, instead of requesting the feasibility of the job completion, in the implementation of Banker's. Such an approach acknowledges – and anticipates – the potential disruption that could be caused in case that resource R_i was not repaired by the time that job j_j requested it.

4.3 Routing Flexibility and the Optimal DAP

Some recent results, presented in [17, 21], identify conditions on the structure of Single-Unit RAS, under which the Optimal DAP is polynomially implementable. The key mechanism enabling these implementations is that, under the conditions stated in [17, 21], every unsafe state of the RAS state space is a deadlock state. While, however, the *state-safety* decision problem is NP-complete [36], the *deadlock detection* problem is polynomially solvable in Single-Unit RAS; an algorithm with complexity $O(m^2\bar{C})$ is provided in [17]. As a result, the Optimal DAP for this sub-classes of Single-Unit RAS can be implemented through one-step lookahead, by addressing the question: *"Is the state resulting from a tentative resource allocation a deadlock?"*, in real-time.

The above control scheme is immediately extensible to *disjunctive* generalizations of these Single-Unit RAS sub-classes, as follows: Every time that we consider advancing a job to its next stage, all possible routing options must be examined, until one is found that leads to a deadlock-free RAS state. Assuming that we have at most K implementational options associated with a single job stage, the complexity of the new test on whether the resulting state is deadlock-free or not, is $O(Km^2\bar{C})$.[14]

Furthermore, similar to the Banker's algorithm case, accommodating the various contingencies occurring during the FMS operation reduces to a detection and recovery problem. In this case, we try to detect and recover from *deadlock* states, arising because of these contingencies. An "Optimal" Recovery problem can be formulated, as well, and the basic scheme of the "Optimal" Recovery algorithm of Definition 6 applies also to this case, with the following modifications: (i) the set $\mathcal{I}$ consists of the jobs j_j which are identified by the deadlock detection algorithm to be entangled in a deadlock, and (ii) the original test $\mathbf{s}(\mathcal{I}\backslash\mathcal{U}^*) \in V_o$ is replaced by the test $\mathbf{s}(\mathcal{I}\backslash\mathcal{U}^*) \notin \mathcal{S}_D$, where $\mathcal{S}_D$ is the set of deadlock states. Computational efficiency improvements, based on pertinent "thinning" of the set $\mathcal{I}$, can be devised for this case, too.

There is one caveat, however. Since the "optimal" policy is polynomially implementable only in certain classes of Single-Unit RAS, every time that a contingency occurs, before anything else, we must test whether the policy applicability conditions are still satisfied. Otherwise, we

[14] The additional factor K in this expression results from the fact that the original test in [17] must be modified, so that a job is declared blocked only after all its possible options for advancement have been examined.

should switch to another, suboptimal DAP.

5 Conclusions

In this paper, we have considered the problem of exploiting the flexibility inherent in contemporary FMS, in order to accommodate various contingencies occurring during the system operation, with minimal disruption. In the proposed formulation, the main constraint is to maintain deadlock-free operation, while disruption is measured by the cumulative "value" of the jobs that might have to be unloaded in order to satisfy the problem constraint. As it was shown in the Introduction, this problem is especially relevant in the operational context of the emerging highly automated semiconductor manufacturing fab. The main problem difficulty arises from the high space complexity, due to the exponentially large number of possible routes that must be considered for every job. As a result, polynomial search-based DAP's are more amenable to integrating dynamic routing schemes, compared to algebraic DAP's, since the latter employ the entire job routes in their policy-defining conditions, and therefore, they require the development of more elaborate computational techniques (cf. the column generation scheme of Section 3.3).

Currently, we are developing the software that implements the computational schemes proposed in this paper. The availability of this code will (i) allow the further statistical evaluation of the computational efficiency of the proposed algorithms, and (ii) the integration of these results to broader real-time FMS control platforms.

Viewing the work of this paper from this broader FMS planning and control perspective, it can be seen that it is an important step towards the development of integrated automated solutions for future FMS's, as it allows the systematic on-line reconfiguration of the system and its controller, in face of unexpected events. Furthermore, the provision of the criticality weights in the problem formulation(s) allows for the association of these reconfiguration decisions to higher level/performance objectives. In fact, the determination of meaningful weight sets for the objective function of these problems is an interesting open problem.

Another open problem along similar lines, is the reconfiguration of the DAP implementations in a way that is *optimal* according to some higher-level performance objectives. For instance, from the description of RO DAP's in Section 3.1, it can be seen that the implementation of this policy is parameterized w.r.t. the imposed resource ordering.[15] Selecting a good ordering for the policy implementation depends on the current system configuration and the overall

[15] Although not apparent from the discussion of Section 3.1, a similar remark applies to RUN DAP.

performance objective(s). The rationalization of these dependencies, and the explicit statement and analytical formulation of the corresponding problem, are additional research topics.

Finally, although we believe that many of these results are extensible to the class of Disjunctive-Conjunctive RAS, the underlying algorithmic details remain to be investigated.

Acknowledgment

The author would like to thank the paper reviewers, whose comments and suggestions led to the clarification of some of the paper ideas, and enhanced its readability.

References

[1] P. F. Drucker. The emerging theory of manufacturing. *Harvard Business Review*, May-June:94–102, 1990.

[2] P. M. Noaker. The search for agile manufacturing. *Manufacturing Engineering*, 113-5:40–43, 1994.

[3] S. B. Joshi, E. G. Mettala, J. S. Smith, and R. A. Wysk. Formal models for control of flexible manufacturing cells: Physical and system models. *IEEE Trans. on Robotics and Automation*, 11:558–570, 1995.

[4] F. F. Suarez, M. A. Cusumano, and C. H. Fine. An empirical study of manufacturing flexibility in printed circuit board assembly. *Operations Research*, 44:223–240, 1997.

[5] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77:81–98, 1989.

[6] S. A. Reveliotis. *Structural Analysis & Control of Flexible Manufacturing Systems with a Performance Perspective.* PhD thesis, University of Illinois, Urbana, IL, 1996.

[7] J. L. Peterson. *Operating System Concepts.* Addison-Wesley, 1981.

[8] Z. A. Banaszak and B. H. Krogh. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Trans. on Robotics and Automation*, 6:724–734, 1990.

[9] N. Viswanadham, Y. Narahari, and T. L. Johnson. Deadlock avoidance in flexible manufacturing systems using petri net models. *IEEE Trans. on Robotics and Automation*, 6:713–722, 1990.

[10] R. A. Wysk, N. S. Yang, and S. Joshi. Detection of deadlocks in flexible manufacturing cells. *IEEE Trans. on Robotics and Automation*, 7:853–859, 1991.

[11] R. A. Wysk, N. S. Yang, and S. Joshi. Resolution of deadlocks in flexible manufacturing systems: Avoidance and recovery approaches. *Journal of Manufacturing Systems*, 13:128–138, 1994.

[12] F. S. Hsieh and S. C. Chang. Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 10:196–209, 1994.

[13] J. Ezpeleta, J. M. Colom, and J. Martinez. A petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on R&A*, 11:173–184, 1995.

[14] K. Barkaoui and I. Ben Abdallah. A deadlock prevention method for a class of fms. In *Proc. of the IEEE Intl Conf. on Systems, Man and Cybernetics*, pages 4119–4124. IEEE, 1995.

[15] M. Lawley, S. Reveliotis, and P. Ferreira. Design guidelines for deadlock handling strategies in flexible manufacturing systems. *Intl. Jrnl. of Flexible Manufacturing Systems*, 9:5–29, 1997.

[16] S. A. Reveliotis and P. M. Ferreira. Deadlock avoidance policies for automated manufacturing cells. *IEEE Trans. on Robotics & Automation*, 12:845–857, 1996.

[17] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira. Polynomial complexity deadlock avoidance policies for sequential resource allocation systems. *IEEE Trans. on Automatic Control*, 42:1344–1357, 1997.

[18] M. Lawley, S. Reveliotis, and P. Ferreira. A correct and scalable deadlock avoidance policy for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation (to appear)*, 1998.

[19] M. Lawley, S. Reveliotis, and P. Ferreira. The application and evaluation of banker's algorithm for deadlock-free buffer space allocation in flexible manufacturing systems. *Intl. Jrnl. of Flexible Manufacturing Systems*, 10:73–100, 1998.

[20] K. Y. Xing, B. S. Hu, and H. X. Chen. Deadlock avoidance policy for petri net modeling of flexible manufacturing systems with shared resources. *IEEE Trans. on Aut. Control*, 41:289–295, 1996.

[21] M. P. Fanti, B. Maione, S. Mascolo, and B. Turchiano. Event-based feedback control for deadlock avoidance in flexible production systems. *IEEE Trans. on Robotics and Automation*, 13:347–363, 1997.

[22] K. Barkaoui and I. Ben Abdallah. Analysis of a resource allocation problem in fms using structure theory of petri nets. In *Proc. of the 1st Intl Workshop on Manufacturing and Petri Nets*, pages 1–15, 1996.

[23] J. Browne, D. Dubois, K. Rathmill, S. P. Sethi, and K. E. Stecke. Classification of flexible manufacturing systems. *The FMS Magazine*, 2, 1984.

[24] A. K. Sethi and S. P. Sethi. Flexibility in manufacturing: A survey. *The International Journal of Flexible Manufacturing Systems*, 2:289–328, 1989.

[25] K. E. Stecke and N. Raman. Fms planning decisions, operating flexibilities, and system performance. *IEEE Trans. on Eng. Management*, 42:82–90, 1995.

[26] C. H. Chung and I. J. Chen. A systematic assessment of the value of flexibility for an fms. In *Proceedings of the 3rd ORSA/TIMS Conference on Flexible Manufacturing Systems: OR models and applications*, pages 27–32. ORSA/TIMS, 1989.

[27] P. H. Brill and M. Mandelbaum. On measures of flexibility in manufacturing systems. *Int. Jrnl. Prod. Res.*, 27:747–756, 1989.

[28] G. K. Hutchinson and D. Sinha. A quantification of the value of flexibility. *Jrnl. of Manufacturing Systems*, 8.

[29] D. Gerwin. Manufacturing flexibility: A strategic perspective. *Management Science*, 39:395–410, 1993.

[30] D. M. Upton. Process range in manufacturing: An empirical study of flexibility. *Management Science*, 43:1079–1092, 1997.

[31] K. E. Stecke and I. Kim. A flexible approach to part type selection in flexible flow systems using part mix ratios. *Int. J. Prod. Res.*, 29:53–75, 1991.

[32] S. A. Reveliotis, M. A. Lawley, and P. M. Ferreira. Structural control policies for flexible manufacturing systems. In *Proceedings of the 11th Intl. Conference on Computer Aided Production Engineering*, pages 255 – 260. Institution of Mechanical Engineers, 1995.

[33] E. R. Boer and T. Murata. Generating basis siphons and traps of petri nets using the sign incidence matrix. *IEEE Trans. on Circuits and Systems-1: Fundamental Theory and Applications*, 41:266–271, 1994.

[34] M. Weiss. Semiconductor factory automation. *Solid State Technology*, pages 89–96, 1996.

[35] P. K. Johri. Practical issues in scheduling and dispatching in semiconductor wafer fabrication. *Journal of Manufacturing Systems*, 12:474–485, 1993.

[36] E. M. Gold. Deadlock prediction: Easy and difficult cases. *SIAM Journal of Computing*, 7:320–336, 1978.

[37] W. L. Winston. *Introduction To Mathematical Programming: Applications and Algorithms, 2nd ed.* Duxbury Press, Belmont, CA, 1995.

[38] A. N. Habermann. Prevention of system deadlocks. *Comm. ACM*, 12:373–377, 1969.