Accommodating FMS Operational Contingencies through Routing Flexibility

Spyros A. Reveliotis

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

Abstract

Overwhelmed by the complexity of the FMS Deadlock Avoidance problem, current research has essentially ignored the aspects (and benefits) related to flexible (dynamic) job routing. Furthermore, extending the currently existing results along this line is nontrivial, primarily because of 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 initiates the investigation on the problem of integrating routing flexibility in FMS structural control, by addressing the problem of "optimal" job re-routing in case of operational contingencies. An analytical problem formulation and an efficient solution algorithm are developed for the case that the FMS is structurally controlled by the Resource Upstream Neighborhood (RUN) policy, a deadlock avoidance policy recently presented in the literature. Beyond the value of the presented results, the paper is intended also to outline a methodological approach to the underlying problems; more extensive results on this research can be found in [1].

1 Introduction

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., [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]). 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 (c.f. [4] and the references cited therein). 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 [5]. 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 FMS deadlock problem have recently appeared in the literature; indicatively, we cite [6, 7, 8, 9, 10, 11, 12, 13, 14]. All this work perceives the manufacturing system as a *resource allocation system* shared by concurrently executing *processes*. Each process is defined as a sequence of *processing stages*, where each stage is 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 underlying resource allocation vectors describing their operation, is provided in [11]. 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 (namely, 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 [11] 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), current research has essentially focused on the first three classes of the above taxonomy, ignoring the effects of routing flexibility in the policy development. Specifically, in the existing work, a *static* routing scheme is assumed, in which the entire routing of a job through the system is predetermined upon its loading. Fixing the future job requirements at this point 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.

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 attempt a first step towards *the integration of deadlock-avoidance control with dynamic routing*. More specifically, we are interested in the development of computationally effective and efficient rerouting policies that exploit the system flexibility in order to accommodate / minimize the disruptive effects of operational contingencies like (i) machine breakdowns and (ii) the arrival of expedient ("hot") jobs. The deadlock avoidance paradigm employed in this work is that of Ω-Deadlock Avoidance Policies (DAP's) applied in Single-Unit RAS [10]. This framework was selected because it has been established to be *provably correct*, *scalable* and *reconfigurable*, and therefore, appropriate to support the real-time control of extensively automated manufacturing systems. Furthermore, the algebraic nature of the Ω-DAP's allows the analytical formulation of the afore-stated rerouting problem. Given the imposed space limitations, in this paper we provide our major results regarding a particular Ω DAP known as the *Resource Upstream Neighborhood (RUN)* policy [10]. More extensive results along similar lines regarding other Ω DAP's, as well as the class of polynomially computable one-step look-ahead DAP's[1] can be found in [1].

The detailed organization of the 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, Ω-DAP's, and the RUN policy. In Section 3, we develop a detailed formulation for the problem of optimal rerouting of the currently running jobs in case of operational contingencies, under RUN DAP. Section 4 proposes a computationally efficient solution to this problem, where emphasis is placed on accommodating the space complexity arising from the combinatorial explosion of the possible job routes. The development of Sections 3 and 4 is elucidated with a small but highlighting example. Finally, the last section concludes the paper presentation, suggesting also a series of enhancements and additional research topics.

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. 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. Hence, 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 cross-product set:

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

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 length of a job stage sequence. 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 asso-

[1] This class of policies is also important since it contains the optimal policy for a large subset of contemporary FMS's [11, 14], as well as an efficient variation of Banker's algorithm for these systems [12].

ciated with a certain job route[2]. 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)* [10], where: (i) An *event* $e \in \mathcal{E}$ of this FSA corresponds to the advancement of any job in the system by one step. (ii) 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. (iii) The automaton *state transition function* $f : \mathcal{S} \times \mathcal{E} \to \mathcal{S}$ 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. (iv) 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. (v) 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.

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 [15].[3] In view of the nontractability of the optimal DAP, Ω DAP's have been proposed in [10] 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 the inclusion of the initial state s^0).

RUN [10]: A particular Ω DAP recently developed in the literature is the *Resource Upstream Neighborhood (RUN)* policy. RUN's defining logic emerges from the 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 *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.*

Given this definition, RUN is expressed by the requirement that *the number of jobs in the upstream neighborhood of a resource never exceeds its buffering capacity.*

Example: We highlight the policy-defining logic and the resource neighborhood construction through an example. Consider a small system consisting of four resources R_1, R_2, R_3, R_4, with a corresponding capacity vector $\mathbf{C} =< 2, 1, 1, 1 >$. Furthermore, suppose that in its current configuration, the system supports the production of three distinct job types, with job routes: $JT_1 :\ R_1 \to R_2 \to R_3,\ JT_2 :\ R_3 \to R_4 \to R_3,\ JT_3 :\ R_1 \to R_2$. 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 array 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. By defining the system *state* $\mathrm{s}(t)$ as a vector with components corresponding to the distinct job stages $JT_{jk},\ j = 1, \ldots, 3,\ k = 1, \ldots, l(j)$, supported by the system in its current configuration, and with $\mathrm{s}_{jk}(t)$ being equal to the number of job instances executing stage JT_{jk} at time step t, it is easy to see that the policy constraints can be expressed by a system of linear inequalities on the system state:, i.e.,

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

[2] 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.

[3] Practically interesting subclasses for which the optimal DAP is polynomially computable through one-step lookahead are provided in [11, 14].

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 The "Optimal" Job Rerouting Problem: Statement and Formulation

Problem Statement: Given the above introduction on RUN logic for FMS structural control under a static routing scheme, a verbal description of the basic problem addressed in this paper is as follows: Consider an FMS configuration, supporting flexible routing, which is currently loaded with 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 $JR_{j,k(j)}$ in that route. Furthermore, suppose that the FMS is structurally controlled by RUN, and hence, the constraints expressed by the corresponding Equation 4 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 the above 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:

$$\max z = \sum_{j,q} x_{jq} \tag{5}$$

s.t.

$$\bar{A}\mathbf{x} \leq \mathbf{C} \tag{6}$$

$$\sum_{q} x_{jq} \leq 1, \quad \forall j \tag{7}$$

$$x_{jq} \in \{0,1\}, \quad \forall j, q \tag{8}$$

The objective function of this formulation states our intention to minimize the number of running jobs unloaded from the system. Constraint 6 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 4, 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.[4] Constraint 7 requires that each job instance is assigned to at most one route. Finally, constraint 8 states the binary nature of our decision variables.

The above formulation covers the problem of the job rerouting in case of resource failing by appropriately updating vector $\mathbf{C}$ 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 executing jobs $\mathcal{I}$, and setting $\sum_q x_{j^*q} = 1$ in the corresponding constraint 7.[5] 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:

$$\max z' = \sum_{j,q} w_j x_{jq} \tag{9}$$

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

4 Solving the "Optimal" Job Rerouting Problem

Formulation (9, 6 – 8) 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 (B&B) [16]. What complicates the problem solution even further is the fact that the length of vector $\mathbf{x}$ – and therefore the number of columns of matrix $\bar{A}$ – is $O(K^{\bar{l}} \sum_i C_i)$. In other words, the computational cost for 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 [16] for handling this problem.

The column generation scheme: Let $\bar{I} \subseteq \mathcal{I}$ denote the set of jobs not assigned 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

[4] It should be easy to see that the column in $\bar{A}$ corresponding to variable x_{jq} is a replication of the A_{RUN} column corresponding to route stage $JR^q_{j,k(j)}$, i.e., the k-th stage of job type JT_j when implemented through the q-th possible route.

[5] A similar formulation addresses the broader problem of whether *any* new job can be loaded into the system. [1] The availability of this decision tool justifies our initial assumption that the currently running jobs $j_j \in \mathcal{I}$ have already been pre-assigned to certain routes.

effect of assigning jobs in $\mathcal{I}\backslash\bar{I}$ to specific routes can be introduced in the formulation by appropriately adjusting the capacity vector $\mathbf{C}$. 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.[6] The proposed column relaxation scheme will first solve optimally the LP relaxation defined by equations 9, 6, 7 and $x_{jq} \geq 0$ on the restriction of $\bar{A}$ on $\mathcal{CR}$. Then, it searches for other possible routes that can improve the current solution, by solving, for each job $j_j \in \bar{I}$, the problem:

$$\begin{aligned}\rho_j &= \min_q\{\lambda_{\bar{A}}^T \bar{A}_{[\cdot,jq]} + \lambda_j - w_j\} \\ &= \min_q\{\lambda_{\bar{A}}^T \bar{A}_{[\cdot,jq]}\} + \lambda_j - w_j \qquad (10)\end{aligned}$$

In Equation 10, $\lambda_{\bar{A}}$ is the projection of the *Simplex Multiplier* vector (*shadow prices* [16]), λ, of the current LP solution, on the constraints generated by Equation 6. λ_j is the Simplex Multiplier corresponding to the Constraint 7 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 B&B node. It remains to define a procedure for solving the subproblem defined by Equation 10.

The column generation algorithm: We compute

$$u = \min_q\{\lambda_{\bar{A}}^T \bar{A}_{[\cdot,jq]}\} \qquad (11)$$

through a *forward dynamic programming (DP)* scheme [16]. 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 $< JT_{j,k(j)}, \ldots, JT_{jp} >$ in which stage JT_{jp} is 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 11, is given by the recursion:*

$$V(R_{jp}^l) = \min_x\{V(R_{jp}^l | R_{j,p-1}^x)\} \qquad (12)$$

where:

$$V(R_{jp}^l | R_{j,p-1}^x) = \begin{cases} V(R_{j,p-1}^x), \text{ if the optimal route to } R_{j,p-1}^x \text{ includes: (i) a resource with capacity higher than } C_{R_{jp}^l}, \text{ or (ii) } R_{jp}^l, \text{ itself;} \\ \min\{V(R_{j,p-1}^x) + \lambda_{R_{jp}^l}, \min_{(t,y):\, C_{R_{j,p-t}^y} > C_{R_{jp}^l}}\{V(R_{j,p-t}^y)\}\}, \text{ o.w.} \end{cases} \qquad (13)$$

Proof: Function $V(R_{jp}^l | R_{j,p-1}^x)$ in Theorem 1 denotes the value of the optimal route that reaches resource R_{jp}^l (i.e., the l-th state of stage JT_{jp}) through resource $R_{j,p-1}^x$ (the x-th state of stage $JT_{j,p-1}$). The key observation for computing this function is that, according to the definition of the resource neighborhoods (c.f., Section 2), *a job instance j_j is in the neighborhood of the resource R_k supporting a subsequent stage, only if the route segment between the current supporting resource (inclusive) and resource R_k does not contain a resource R_d with $C_{R_d} > C_{R_k}$.* Hence, the first branch of Equation 13 follows immediately from this remark. For the second branch, notice that its defining condition implies that if resource R_{jp}^l is reached through the optimal route to resource $R_{j,p-1}^x$, job j_j must be included in the neighborhood of R_{jp}^l. Therefore, the value of the resulting route is $V(R_{j,p-1}^x) + \lambda_{R_{jp}^l}$, where $\lambda_{R_{jp}^l}$ is the shadow price of the RUN constraint corresponding to resource R_{jp}^l. However, if resource $R_{j,p-1}^x$ is reached through a resource $R_{j,p-t}^y$ with capacity $C_{R_{j,p-t}^y} > C_{R_{jp}^l}$, then $V(R_{jp}^l | R_{j,p-1}^x, R_{j,p-t}^y) = V(R_{j,p-1}^x | R_{j,p-t}^y) = V(R_{j,p-t}^y)$. The last equality holds since, by the defining assumption of the second branch in Equation 13, the optimal route to $R_{j,p-1}^x$ provides a feasible route segment with resources of capacity lower than or equal to $C_{R_{jp}^l}$, and therefore, strictly lower than $C_{R_{j,p-t}^y}$. Since it is possible that $V(R_{j,p-1}^x) < V(R_{j,p-t}^y)$ but $V(R_{j,p-1}^x) + \lambda_{R_{jp}^l} > V(R_{j,p-t}^y)$, we must also consider these alternative routes to R_{jp}^l through $R_{j,p-1}^x$ in the computation of $V(R_{jp}^l | R_{j,p-1}^x)$; this is the content of the internal min{} in the second branch of Equation 13. □

The complexity of this DP scheme is $O(\bar{l}^2 K^2)$.

Example: We demonstrate the application of the proposed scheme, by extending the example of Section 2. Specifically, consider that the system described in that example is currently 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

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

job instance j_1 is infeasible. 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 job stage JT_{22}. Hence, job instance j_1 has four routing options for its remaining processing, i.e., $o_{11}: R_2 \rightarrow R_3$, $o_{12}: R_2 \rightarrow R_1$, $o_{13}: R_4 \rightarrow R_3$, $o_{14}: R_4 \rightarrow R_1$. Similarly, 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 of the "optimal" job rerouting problem (c.f. Section 3, 5– 8) for this small example is:

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

s.t.

$$\begin{bmatrix} & 1 & & 1 & & 1 \\ 1 & 1 & & & & \\ 1 & & 1 & & 1 & \\ & & 1 & 1 & 1 & \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \end{bmatrix} \leq \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{aligned} x_{11} + x_{12} + x_{13} + x_{14} &\leq 1 \\ x_{21} + x_{22} &\leq 1 \\ x_{ij} \in \{0,1\} \quad \forall i,j & \end{aligned} \tag{14}$$

Furthermore, it is easy to see, by inspection, that the optimal route assignment 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 proposed column generation scheme, 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.

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

$$\begin{aligned} x_{11} &\leq 1 \\ x_{21} &\leq 1 \\ 0 \leq x_{ij} \leq 1 \quad \forall i,j & \end{aligned} \tag{15}$$

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:

State	Stage 1	Stage 2
1	R_2	R_3
2	R_4	R_1

The optimal value function for the above shadow prices, λ^1, is: $V_1^1(1,1) = V_1^1(1,2) = 0$, $V_1^1(2,1) = \min\{V_1^1(2,1|1,1) + \lambda_{R_3}^1, V_1^1(2,1|1,2) + \lambda_{R_3}^1\} = 1$, $V_1^1(2,2) = \min\{V_1^1(2,2|1,1) + \lambda_{R_1}^1, V_1^1(2,2|1,2) + \lambda_{R_1}^1\} = 0$. 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:

State	Stage 1	Stage 2
1	R_4	R_3
2	R_1	–

and the optimal value function is: $V_2^1(1,1) = V_2^1(1,2) = 0$, $V_2^1(2,1) = \min\{V^1(2,1|1,1) + \lambda_{R_3}^1, V^1(2,1|1,2)\} = \min\{1,0\} = 0$. 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.

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

$$\begin{aligned} x_{11} + x_{12} + x_{14} &\leq 1 \\ x_{21} + x_{22} &\leq 1 \\ 0 \leq x_{ij} \leq 1 \quad \forall i,j & \end{aligned} \tag{16}$$

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 $\rho_1^2 = u_1^2 + \lambda_1^2 - w_1 = 0 + 1 - 1 = 0$, $\rho_2^2 = u_2^2 + \lambda_2^2 - w_2 = 0 + 1 - 1 = 0$. Hence, 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 $R_1 \rightarrow R_3$. Notice that in this approach, route $o_{13}: R_4 \rightarrow R_3$ was never explicitly considered in the LP formulations.

5 Discussion

The previous example has demonstrated the kind of computational economies obtained through the computational scheme proposed in this paper. It is expected that for larger size IP's, – i.e., larger number of job instances with many more possible routes – these economies will be much more significant, driving the search to focus on "good" routes. Also, initial experimentation with these formulations has indicated that the underlying LP-relaxations, even though not presenting the integrality property, are well-behaved (e.g., in the previous example, all three formulations of Equations 14– 16 had integral optimal solutions). Further encoding of the entire computational scheme will allow the experimentation with cases of larger size, and the quantification of the expected computational gains. Currently, we are planning the integration of this re-routing tool to a broader framework concerning the real-time control of large-scale flexibly automated discrete-part manufacturing systems. As an example, in this framework it might be possible to rationalize the computation of the weight-values to be used in the objective function of Equation 9, by pertinent use of performance-oriented data and models. Moreover, in [10], it is shown that if the neighborhood construction is based on any other ordering of the system resources, rather than the one induced by the resource capacities, the resulting RUN implementation is still correct. This result implies the existence of an entire set (family) of RUN implementations for a given system configuration, each presenting different levels of efficiency. In the context of the "optimal" re-routing problem, this remark implies that beyond the search for a good set of routes, another (orthogonal) search can be performed in the space of the resource orderings (i.e., resource permutations) for one that leads to the least restrictive RUN implementation for the current job load. Hence, a two-stage search can be organized, where the higher-level search is carried over the space of resource orderings, and uses the "optimal" rerouting problem presented in this paper to evaluate a given resource permutation. The computational details and the efficiency gains obtained by such an extended scheme is part of our future research agenda. Concluding the paper discussion, we want to emphasize that the results reported herein are part of a broader effort to develop effective (provably correct) and (performance and computationally) efficient models and operational procedures for the problem of real-time control of flexibly automated, reconfigurable discrete-part manufacturing systems.

References

[1] S. A. Reveliotis. Accommodating fms operational contingencies through routing flexibility. Technical Report (submitted to IEEE Trans. on R&A), School of Industrial & Systems Eng., Georgia Tech, 1997.

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

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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 (to appear).*

[13] 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.

[14] 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.

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

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