Polynomial-Complexity Deadlock Avoidance Policies for Sequential Resource Allocation Systems

Spiridon A. Reveliotis
School of Industrial & Systems Engineering
Georgia Institute of Technology
765 Ferst Drive
Atlanta, GA 30332
Tel #: (404) 894-6608
E-mail: *spyros@isye.gatech.edu*

Mark A. Lawley
Department of Industrial Engineering
The University of Alabama
Box 870288
Tuscaloosa, AL 35487
Tel #: (205) 348-3337
FAX: (205) 348-8573
E-mail: *MALAWLEY@coe.eng.ua.edu*

Placid M. Ferreira*†
Department of Mechanical and Industrial Engineering
University of Illinois
361 Computer & Systems Research Laboratory
1308 W. Main Str.
Urbana, IL 61801
Tel #: (217) 333-0639
FAX: (217) 244-9956
E-mail: *placid@uxh.cso.uiuc.edu*

Abstract

The development of efficient deadlock avoidance policies (DAP's) for sequential resource allocation systems (RAS) is a problem of increasing interest in the scientific community, largely because of its relevance to the design of large-scale flexibly automated manufacturing systems. Much of the work on this problem existing in the literature, is focused on the so called Single-Unit RAS model, which is the simplest model in the considered class of RAS. Furthermore, due to a well-established result stating that, even for single-unit RAS, the computation of the maximally permissive DAP is computationally intractable (NP-hard), many researchers (including our group) have focused on obtaining good suboptimal policies which are computationally tractable (scalable) and provably correct. In the first part of the paper, it is shown, however, that for a large subset (in fact, a majority) of single-unit RAS, the optimal DAP can be obtained in real-time with a computational cost which is a polynomial function of the system size (i.e., the number of resource types and the distinct route stages of the processes running through the system). The implications of this result for the entire class of single-unit RAS are also explored. With a result on the design of optimal DAP's for single-unit RAS, the second part of the paper concentrates on the development of scalable and provably correct DAP's for the more general case of conjunctive RAS.

*Corresponding author

†This work was partially funded by the NSF grant NSF-DDM-91-57191

1 Introduction

In this paper we revisit the problem of *deadlock avoidance* in *sequential resource allocation systems*. A *resource allocation system (RAS)* consists of a set of concurrently running processes which, at certain phases of their execution, require the exclusive use of a number of the system resources to successfully run to completion [16]. The resources are in limited supply, and they are characterized as *reusable*, since their allocation and deallocation to requesting processes affects neither their nature nor their quantity. Furthermore, the entire resource allocation system is characterized as *sequential* because it is assumed that every process, during its execution, undergoes a *predefined* sequence of resource allocation and deallocation steps. Specifically, for the purposes of our study, every process can be described by a sequence of stages, with every stage defined by the subset of resources required for its successful execution.[1]

The study of resource allocation models was initiated in the field of computer system engineering in the late '60s and early '70s, in the context of a broader effort in providing robust and efficient operating systems [6, 9, 10, 11]. It was quickly realized and formally established [5, 11] that most of the relevant resource allocation environments give rise to situations in which a number of processes get into a "circular wait" state where they wait for each other to release resources required for the execution of their next processing stages; this situation was defined as a system *deadlock*. Effective handling of the deadlock arising in the various resource allocation environments has turned out to be a major concern in the operation of these systems.

In general, three major strategies were identified as applicable to deadlock-handling [16]: (i) *Prevention*, or the design of *off-line* resource requesting protocols for processes to rigorously establish that the resulting operation is deadlock-free, (ii) *Detection & Recovery*, or the development of algorithms that can *detect* a deadlock and provide corrective mechanisms that can *roll* the system *back* to a deadlock-free state, typically by preempting processes, and (iii) *Avoidance*, or the development of *on-line* control policies that use *feedback* information on the current resource allocation status, possibly combined with information about future process resource requirements, to *keep* the system *away* from deadlock states. This work focuses on the third approach, i.e., deadlock avoidance. Avoidance strategies are not as conservative as prevention, and they are particularly useful in computer control of physical systems where deadlock detection & resolution techniques require manual intervention and are typically prohibitively expensive, e.g., a flexible manufacturing system or a mono-rail complex.

The work of the early period set the foundation for understanding the nature of the deadlock

[1] A more detailed discussion of the modeling assumptions and a classification of the resulting resource allocation system models will be given in a later section of the paper.

problem. All of the solutions proposed to date are strongly motivated by the general guidelines suggested in this pioneering work. For instance, Dijkstra's *Banker Algorithm* [6] is still the most fundamental approach to deadlock avoidance. A primary characteristic of this work, though, is that it was oriented to the resource allocation systems motivated by the computer system environments. Typically, these environments lack any a priori information about the subsequent requirements of the involved processes. As a result, all of the proposed solutions were essentially addressing the *worst-case scenario*, i.e., the safety of satisfying a pending request under the assumption that every process immediately posts its (prespecified) maximal possible request for every resource type. It is obvious that such an approach neglects the *sequential* characteristics of the process needs, leading to more restrictive system behavior.

The problem of deadlock avoidance policy design under complete knowledge of the process allocation sequences was more thoroughly studied towards late '70s, with emphasis placed on its computational complexity. Specifically it was shown in [2, 8] that, in its general definition, the problem of determining the *safety* of a given resource allocation state – i.e., whether there exists a feasible allocation sequence that can lead to the successful completion of all processes in the system – is in the class of *NP-complete* problems. The primary implication of this finding is that even under perfect a priori process information, the computation of the *optimal*, or maximally permissive, avoidance policy is practically impossible. However, it was also shown in the same papers [2, 8], that when the resource allocation system presented some further properties w.r.t. the resource nature and the process sequences, the optimal avoidance policy is polynomially computable. This has led researchers in the field to believe that it might be possible to develop more efficient policies even for the cases that are known to possess an NP-complete optimal solution, by appropriately exploiting the allocation sequence information.

Recently, these "hunches" have obtained a more practical significance, since the technological innovations in the rapidly expanding field of *Flexible Automation and Manufacturing Systems* have set the motivational environment for the study of the deadlock avoidance problem for sequential resource allocation systems, i.e., in the case that the evolution of the process allocation requests is a priori perfectly known. In our past work, communicated in [13, 21], it was found that: (i) a large volume of the published research has failed to deal effectively with the nonpolynomial complexity of the underlying problem, yet, (ii) it is possible to design *scalable* (i.e., of polynomial complexity w.r.t. the number of system resources and process stages) and *provably correct* avoidance policies which are less restrictive than their worst-case scenario-based counterparts (e.g., [3, 21, 23]).

Furthermore, it was found [4, 13] that the tractability of the resulting problem and the features of the corresponding solution significantly depend on the structure of the resource requests posed by the RAS processes. This has led to the classification of the different RAS on the basis of the valid resource requests posted by the processes in the system [4, 13]. The most significant classes identified in the resulting taxonomy are: (i) the *Single-Unit* RAS, where every process stage requires only *one* unit from a *single* resource for its successful execution, (ii) The *Single-Type* RAS, where every process stage requires an *arbitrary* number of units, however, all of the *same* resource type, for its successful execution, (iii) The *Conjunctive (AND)* RAS, where every process stage, to be successfully executed, requires the simultaneous exclusive service of an arbitrary number of units of different resource types, (iv) The *Disjunctive/Conjunctive (AND/OR)* RAS, where every stage is associated with a set of conjunctive requests, the implication being that satisfaction of any of these requests by the system is sufficient for the successful execution of the step.

To provide a more intuitive interpretation of these class definitions, notice that in the discrete-part manufacturing context, which is the primary motivation for our work, the Single-Unit RAS class corresponds to the case in which all required auxiliary resources, like fixtures and tools, are allocated a priori to the system workstations, and thus, the only shared resource required by job at any processing stage is one unit of *buffering* capacity on the supporting workstation or the material handling system. Being a physical entity, this is the minimal requirement posted by any job in the system. Single-type RAS models also correspond to situations in which the only shared resource is the limited buffering capacity of the system components; however, this model covers also the case in which jobs are organized as tightly connected *batches*, and therefore, the buffering requirements will vary depending on the number of jobs in the batch. Conjunctive RAS cover the more general case in which jobs contest not only for buffering space, but also for other *auxiliary* resources which are in limited supply, and therefore, they can give rise to deadlocks, too. Finally, the Disjunctive/Conjunctive model is intended to cover environments in which the increased operational flexibility implemented in the system workstations allows the on-line re-routing of the jobs (*routing flexibility*).

Higher-numbered members of the above taxonomy subsume those numbered lower, but the models get more complicated. Hence, even though the problem of polynomial-complexity deadlock avoidance in sequential models was posed in its general version in [13], the initiating analysis has been restricted in the single-unit RAS case. The work presented in this paper has two objectives:

1. First, it is shown that the most broadly studied problem of deadlock avoidance in the Single-Unit RAS can be resolved *optimally* (!) in polynomial time when all system re-

source types are available in *more than one* units. Given the above discussion on the taxonomy interpretation in the discrete-part manufacturing context, it follows, then, that in any manufacturing system modeled by a Single-Unit RAS, all that is required in order to resolve deadlocks optimally in polynomial time, is to ensure that every system component (i.e., workstation and/or material handling system) has a buffering capacity to simultaneously accommodate at least two jobs. Furthermore, for Single-Unit RAS in which unit-capacity resources are unavoidable,[2] an efficient control scheme is obtained by decomposing the original system into two subsystems: (i) that consisting of all unit-capacity resources of the original system, and (ii) the remaining resources, with non-unit capacity (i.e., equal to or higher than two).

2. Having conclusively resolved the deadlock avoidance problem in the single-unit RAS case, allows us to concentrate, in the second part of the paper, on the most general case of the *conjunctive (AND)* RAS model. To this end, the system modeling and the policies developed in [21, 23] are *generalized* so that they apply to these models. Notice, however, that even though the fundamental problem structure remains the same, the technical details of the analysis and the policy (re-)definitions are significantly different. To a large extent, this results from the fact that in the Single-Unit RAS, process stages can be uniquely mapped to their supporting resource, while this is not the case for the Conjunctive RAS model. Furthermore, in the Single-Unit RAS model, deadlocked resources must be completely filled to capacity, which gives a very intuitive basis for the problem analysis. Hence, the undertaken analysis and policy design for Single-Unit RAS have employed a resource-based point of view. This perspective is not functional for the conjunctive RAS model. Thus, to extend our intuition and previous results to this more complicated model, a different process stage-based perspective is employed. However, the intuition and insight obtained through the investigation of the Single-Unit RAS case, were extremely valuable in the formulation of these more general policies. Finally, as it is expected, the generalized results, when restricted to the Single-Unit RAS context, simplify to those originally developed in our past work.

The detailed paper structure is as follows: Section 2 provides a review of the formal RAS modeling developed in [21], with the various system concepts and properties redefined and generalized so that they pertain to the broader class of conjunctive RAS. Section 3 communicates our new result obtained for the single-unit RAS case and discusses its practical implications,

[2] This might be, for instance, an existing mono-rail system with some single-rail segments.

while Section 4 deals with the problem of polynomial-complexity deadlock avoidance in conjunctive RAS models. Two policies, RUN and RO, initially developed in [21, 23] for single-unit RAS, are generalized so that they apply to these more general models. Finally, Section 5 draws conclusions and discusses some other research issues related to the control of the considered RAS.

2 Conjunctive Resource Allocation Systems and their Deadlock Properties

This section is a revision of the basic concepts and properties of a sequential Resource Allocation System (RAS) model, originally developed in Sections 2 – 4 of [21] for the single-unit RAS case. However, all the relevant notions are redefined and generalized, so that they apply to the *Conjunctive (AND)* model of the taxonomy presented in [13]. More specifically, as it was stated in the introduction, the more general and complicated structure of the Conjunctive RAS model, as compared to its Single-Unit counterpart, has rendered invalid the resource-based analysis and policy design pursued in [21]. Instead, processing stage-based modeling and analysis is applied. Therefore, detailed redefinition of most of the underlying concepts is inevitable. To provide these definitions without compromising the paper integrity, we have opted to include the entire set of them, even though some are a simple restatement of their counterparts in [21]. Yet, the interested reader is referred to [21] for a more extensive and intuitive introduction of the basic problem structure and its underlying concepts.

2.1 The RAS model

The considered RAS model is defined by a number of *resource types*, denoted by $\mathcal{R} = \{R_i,\ i = 1,\ldots,m\}$, and a number of *process types*, denoted by $\mathcal{J} = \{JT_j,\ j = 1,\ldots,n\}$. Every resource R_i is further characterized by its *capacity* C_i, i.e., a finite positive integer indicating how many units of resource type R_i the RAS possesses. Process type JT_j is defined by a sequence of m-dimensional vectors $< JT_{jk},\ k = 1,\ldots,l(j) >$, with component $JT_{jk}[q],\ q = 1,\ldots,m$, indicating how many units of resource R_q are required for the successful completion of the k-th step of the j-th process type.[3] The sequence of m-dimensional vectors defining process type $JT_j,\ j = 1,\ldots,n$, will be referred to as its *route*, while the distinct elements of the sequence

[3]Notice that for all practical purposes, it must be $\sum_q JT_{jk} > 0,\ \forall j,k$. A stage for which $\sum_q JT_{jk} = 0$ can be considered as a breakpoint at which the process is temporarily unloaded from the resource allocation system, and the corresponding process route can be split into two different segments.

will be characterized as the *route stages*.[4] Given this formalization of the system description, the system state is defined as follows:

Definition 1 *The RAS* state $\mathrm{s}(t)$ *at time t is a vector of dimensionality $D = \sum_{j=1}^{n} l(j)$ – i.e., equal to the total number of distinct route stages in the system –, with component $\mathrm{s}_i(t)$, $i = 1, \ldots, D$, being equal to the number of processes executing step k of route JT_q at time t, where q is the largest integer s.t. $i > \sum_{j=1}^{q-1} l(j)$ and $k = i - \sum_{j=1}^{q-1} l(j)$.*

Notice that the information contained in the RAS state is sufficient for the determination of the distribution of the resource units to the various processes, as well as the *slack* (or idle) capacity of the system. Furthermore, the finiteness of the resource capacities implies that the set of distinct RAS states is finite; let it be denoted by $\mathcal{S} = \{s^i,\ i = 0, 1, \ldots, Z\}$.

The system state changes in one of the following three ways: (i) *loading* a new process in the system, (ii) *advancing* an already loaded process to its next route stage, and (iii) *unloading* a finished process. During a single state transition only one process can proceed. The resulting step, however, is *feasible* only if the (additionally) required resources can be obtained from the system slack capacity. Let $\mathcal{F}(s^i)$ denote the set of feasible steps that can be taken in state s^i. Assuming that transitions corresponding to feasible steps are occurring instantaneously, the operation of the considered RAS can be more formally described by a *finite state automaton (FSA)* [12, 17]. The *event set E* of this automaton consists of all the possible route steps executed in the RAS, its *state set S* is identified by the set of RAS states, the *state transition function f : $S \times E \to S$* is a formal expression of the aforementioned feasibility requirements of the RAS operation, the *initial* and *final states* correspond to the state s^0 in which the RAS is idle and empty of any processes, while the *language L* accepted by the automaton consists

[4] In the context of this formalism, the RAS taxonomy, originally presented in [13] and discussed in the introductory section, can be restated by appropriately constraining the valid resource request vectors defining the process stages. Specifically,

1. the *conjunctive (AND)* RAS model allows the request vector components to take any integer value from 0 to the corresponding resource capacity,
2. the *single-type* RAS model results from the conjunctive RAS model by imposing the additional constraint that only one of the request vector components is nonzero,
3. the *single-unit* RAS model is the single-type model with the extra constraint that the nonzero component can take only the value of 1, and finally,
4. the *disjunctive/conjunctive (OR/AND)* RAS model defines for every process stage a set of conjunctive resource request vectors; the model is characterized as disjunctive since the stage can be successfully executed once the process is allocated any of the conjunctive requests in this set.

of those event sequences that correspond to complete runs of the system. Finally, notice that, as it has already been stated above, the resulting RAS model corresponds to the *conjunctive (AND)* case of the taxonomy presented in [13].

The following definitions, stated in the mathematical formalism of the underlying FSA model, provide a rigorous framework for studying the problem of deadlock in the considered RAS.

Definition 2 *In the conjunctive RAS model,*

1. *State s^i is* reachable from *state s^j, denoted $s^i \leftarrow s^j$, iff there exists a sequence of events that can bring the system from state s^j to state s^i. In the FSA notation,*

$$\forall s^i, s^j \in \mathcal{S},\ s^i \leftarrow s^j \Longleftrightarrow \exists u \in E^* :\ f(s^j, u) = s^i$$

Furthermore, a state $s^i \in S$ is a reachable state, *iff $s^i \leftarrow s^0$. The set of reachable (unreachable) states will be denoted by S_r $(S_{\bar{r}})$.*

2. *State s^i is a* safe *state iff state s^0 is reachable from state s^i. A state which is not safe will be called an* unsafe *state. In the FSA notation,*

$$\forall s^i \in \mathcal{S},\ \mathtt{safe}(s^i) \Longleftrightarrow \exists u \in E^* :\ f(s^i, u) = s^0 \Longleftrightarrow s^0 \leftarrow s^i$$

The set of safe (unsafe) states will be denoted by S_s $(S_{\bar{s}})$.

3. *A state transition emanating from a safe state is* safe *iff it results in a safe state. In the FSA notation,*

$$\forall s^i \in \mathcal{S}_s,\ \forall e \in \mathcal{F}(s^i),\ \mathtt{safe}(e/s^i) \Longleftrightarrow f(s^i, e) \in \mathcal{S}_s$$

Similar to the single-unit RAS, the natural reason for the existence of unsafe states is *deadlock* [21]. An appropriate definition for deadlock in the conjunctive RAS model is as follows:

Definition 3 *Define the* apparent slack *of resource R_j w.r.t. a set of processes $\mathcal{DP}^i$ executing in the system, to be equal to the capacity of R_j, C_j, minus the number of its units allocated to processes in $\mathcal{DP}^i$.*

Then, state s^i is a partial deadlock, *if there exists a (sub)set of processes $\mathcal{DP}^i$ executing in the system, s.t. for each process $j_j \in \mathcal{DP}^i$, there exists a resource R_k, s.t. j_j is requesting a number of units of it, which is larger than the apparent slack of R_k w.r.t. set $\mathcal{DP}^i$.*

This definition can also be used for the single-unit RAS case. However, the fact that in this particular model every process stage is associated with a single resource, allows the analysis of the deadlock avoidance problem from a resource-based perspective. In our past work with single-unit RAS, we have found this view point very instrumental for developing effective and efficient algorithms for single-unit RAS. Furthermore, this perspective is used in Section 3 for the development of the proof of the new optimality result pertaining to these systems. Therefore, this specialized definition of deadlock in single-unit RAS is also included here [21]:

Definition 4 *In a single-unit RAS, state s^i is a* partial deadlock*, if a subset of its resources, $\mathcal{DR}^i$, is filled to capacity, and every process holding a unit of these resources requires transfer to another resource in $\mathcal{DR}^i$ for the execution of its next stage.*

Returning to the more general definition of deadlock, – i.e., Definition 3 – we discuss next an algebraic characterization of the state safety problem for the conjunctive RAS case, which is based on this broader definition:

Consider the RAS state $s^i = \mathsf{s}_0$ where J_0 processes, $\{j_1, j_2, \ldots, j_{J_0}\}$, are loaded in the system. Let U_i^0 denote the stage executed by process j_i in s_0. Since a state transition in the considered RAS model concerns the advancement of a single process by one step, it follows that, if state s_0 is safe, the total number of transitions required for running all the currently loaded processes to completion, without loading any additional processes, is $T_t = \sum_{i=1}^{J_0}[l(j_i) + 1 - U_i^0]$, where $l(j_i)$ denotes the total number of stages in the route executed by process j_i.[5] Finally, let $\{\delta_{ikt}\}$ denote a set of *binary* variables with

$$\delta_{ikt} = \begin{cases} 1 & \text{if process } j_i \text{ executes its } k\text{-th stage at step } t \\ 0 & \text{o.w.} \end{cases}$$

where $i \in \{1, \ldots, J_0\}$, $k \in \{U_i^0, \ldots, l(j_i) + 1\}$ and $t \in \{0, \ldots, T_t\}$.

State s_0 is safe iff the following system in variables δ_{ikt} is feasible:

$$\delta_{ik0} = \begin{cases} 1 & \text{if } k = U_i^0 \\ 0 & \text{o.w.} \end{cases} \quad \forall i, k \tag{1}$$

$$\sum_{i=1}^{J_0} \sum_{k=U_i^0}^{l(j_i)+1} JT_{j_ik}[r]\delta_{ikt} \leq C_r, \quad \forall R_r \in \mathcal{R}, t \tag{2}$$

$$\sum_{k=U_i^0}^{l(j_i)+1} \delta_{ikt} = 1, \quad \forall i, t \tag{3}$$

[5] The $l(j_i) + 1$ step corresponds to the unloading of the process.

$$\sum_{t=0}^{T_t} \delta_{i,l(j_i)+1,t} \geq 1, \quad \forall i \tag{4}$$

$$\delta_{ikt} - (\delta_{ik(t+1)} + \delta_{i,k+1,(t+1)}) \leq 0, \quad \forall i, k(\neq l(j_i)+1), t(\neq T_t) \tag{5}$$

$$(\delta_{ik(t+1)} - \delta_{ikt}) JT_{ik}[r] + \sum_{j=1}^{J_0} \sum_{l=U_j^0}^{l(j_j)} JT_{jl}[r]\delta_{jlt} \leq C_r, \quad \forall i, k(\neq l(j_i)+1), t(\neq T_t), R_r \in \mathcal{R} \tag{6}$$

$$\delta_{ikt} \in \{0,1\}, \quad \forall i, k, t \tag{7}$$

Equation 1 models the description of the initial state s_0, and it represents the set of initial conditions. Equation 2 states that the maximal number of units that can be allocated from a resource type is limited by its capacity. Equation 3 expresses the fact that every process instance is accounted for at a certain processing stage or has completed processing at any time step. Equation 4 simply states that for a RAS state to be safe, it is necessary that every process is eventually unloaded from the system over the considered horizon. Equation 5 represents the precedence constraints introduced by the process routes, i.e., every movement of a process in the RAS must obey its routing sequence. Equation 6 imposes the requirement that a process can proceed to its next step only if the additionally required resource units are available from the system slack capacity. Finally, Equation 7 states the binary nature of the δ variables.

We shall return to this system of diophantine equations in Section 4.1, where we discuss its role in developing correct and scalable DAP's for the considered RAS classes.

2.2 Deadlock Avoidance Policies – General Definitions

Broadly speaking, a *Deadlock Avoidance Policy (DAP)* tries to restrict the operation of a RAS to its reachable and safe subspace S_{rs}, by *disabling* an appropriate set of feasible transitions so that the unsafe subspace $S_{r\bar{s}}$ is rendered unreachable from state s^0. At the same time, it must be ensured that every state s^i in the remaining graph (i.e., *reachable under the control policy*) is still safe (i.e., there exists a directed path *in the remaining graph* leading from state s^i to s^0). States which are reachable under the DAP and from which progress is inhibited by the policy-imposed constraints and not by the RAS structure, are characterized as *restricted* deadlocks in the deadlock literature [3].

These ideas are formalized by the following definitions in the FSA context:

1. An *avoidance policy* $\mathcal{P}$ is a function

$$\mathcal{P}:\ \mathcal{S} \rightarrow 2^E,\ \ \mathcal{P}(s^i) = \{e \in \mathcal{F}(s^i):\ e \text{ is selected by the policy}\}$$

   Events $e \in \bigcup_i \mathcal{P}(s^i)$ are called the *(policy-)enabled* events.

2. Given an avoidance policy $\mathcal{P}$, let $s^i \overset{\mathcal{P}}{\leftarrow} s^j$ denote the fact that state s^i is reachable from state s^j through an event sequence which comprises policy-enabled events only. Let $S_r(\mathcal{P}) = \{s^i :\ s^i \overset{\mathcal{P}}{\leftarrow} s^0\}$ and $S_s(\mathcal{P}) = \{s^i :\ s^0 \overset{\mathcal{P}}{\leftarrow} s^i\}$. Then, policy $\mathcal{P}$ is *correct* iff $S_r(\mathcal{P}) \subseteq S_s(\mathcal{P})$. (This is equivalent to the requirement that the policy is deadlock and restricted deadlock-free.)

3. The correct avoidance policy $\mathcal{P}^*$ is *optimal* or *maximally permissive* iff

$$\forall s^i \in \mathcal{S}_{rs},\ \forall e \in \mathcal{F}(s^i),\ e \in \mathcal{P}^*(s^i) \Longleftrightarrow f(s^i, e) \in \mathcal{S}_s$$

Notice that the above characterization of the optimal policy has the following three implications: (i) For a given RAS, the optimal avoidance policy $\mathcal{P}^*$ is *unique*. (ii) $\mathcal{S}_r(\mathcal{P}^*) = \mathcal{S}_{rs}$. Establishing the optimal control policy $\mathcal{P}^*$ is equivalent to removing from the reachability graph those transition arcs that belong to the *cut* $[\mathcal{S}_{rs}, \mathcal{S}_{r\bar{s}}]$. (iii) In [2, 8] it is shown that, in the general case, the problem of determining the safety of a RAS state is NP-complete. Since the inclusion of a transition to the optimal avoidance policy $\mathcal{P}^*$ depends on the safety of the successor state, it follows that obtaining policy $\mathcal{P}^*$ is an NP-hard problem [7].

3 DAP Optimality in the Single-Unit RAS model

The concluding remark of the previous section about the complexity of computing the optimal policy for general sequential RAS, led many researchers, including our group, to focus their efforts in obtaining good *suboptimal* policies, i.e., policies which are computationally tractable and still are relatively efficient w.r.t. the overall system performance. It has also already been mentioned that most of these initiating efforts (e.g., [3, 13, 15, 21, 25]) focused on single-unit RAS, since this model captures most of the relevant features of the deadlock avoidance problem in a sequential RAS, while still being the most tractable model in the [13] taxonomy. However, in this section, we present a new result, which establishes that for a large subclass (in fact, a majority) of single-unit RAS, the deadlock avoidance problem can be resolved *optimally* in polynomial time. Furthermore, it is shown in the ensuing discussion that this result has very significant practical implications for the problem of deadlock avoidance in the discrete-part manufacturing context.

The formal statement and proof of this result is as follows:

Claim 1 *In a single-unit RAS where every resource R_i, $i = 1, \ldots, m$, has capacity $C_i \geq 2$, every unsafe state is a partial deadlock state.*

Proof: Before proceeding with the most rigorous development of the proof, it is mentioned that Figure 1 is intended as a visualizing aid of its most important points. The proof itself is based on the following contradiction argument:

Suppose that there exist unsafe states which are *not* partial deadlocks. Then, the following series of observations holds.

(i) First, notice that the definition of state-safety essentially requires that all currently loaded processes can run to completion. It is therefore legitimate, while reasoning about the safety or unsafety of a RAS state, to assume that no new processes are loaded in the system during the evolution of the system operation.

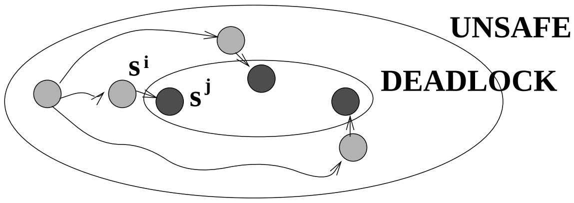
(ii) Since every unsafe state leads unavoidably to a (partial) deadlock, there exists a state s^i that is not a deadlock state itself, but exactly one transition away from a (partial) deadlock. In other words, further advancement of any process in that system state s^i leads to a deadlock. Hence, in state s^i, there must exist:

a. A non-empty subset $\mathcal{H}$ of resources that are filled to capacity and are not satisfying Definition 4 (for if otherwise, a deadlock cannot be produced in a single transition from s^i, since it requires at least two resources filled to capacity, and a single transition can add at most one member to this set).

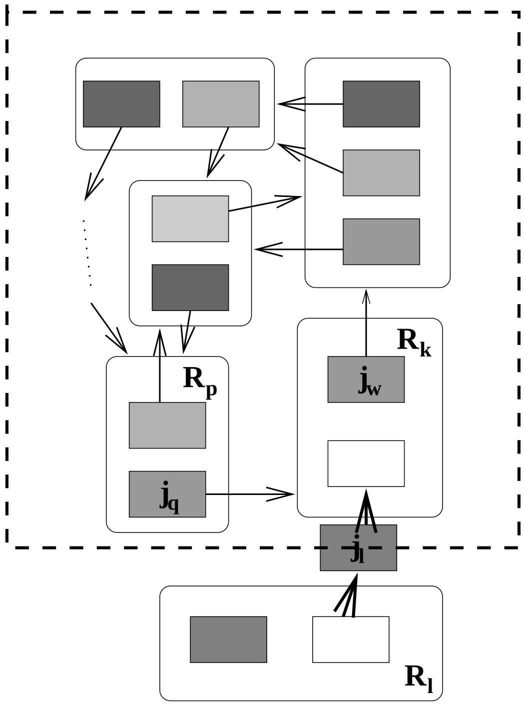
b. A non-empty set $\mathcal{G}$ of resources with exactly one unit of free (slack) capacity. For if $\mathcal{G}$ is empty, no new resource can be added to $\mathcal{H}$ in a single transition from s^i, and therefore, no (new) deadlock can be thus created. Further, since s^i is exactly one step away from a partial deadlock, a member of $\mathcal{G}$ either produces a deadlock upon allocating its unit of free capacity, or it has no pending requests in state s^i.

c. Finally, all pending requests in state s^i must necessarily be for resources in $\mathcal{H}$ (and hence they cannot be immediately granted) or for members of $\mathcal{G}$ (and should result in a deadlock). Otherwise, state s^i is not unavoidably a single transition away from deadlock.

(iii) Let us consider a member of $\mathcal{G}$, say R_k, whose unit of free capacity has pending requests. According to (ii), granting of any pending request for R_k results in state s^j, which is a partial deadlock. This deadlock must involve R_k, since it is the only addition to set $\mathcal{H}$, and,



(a) The topological relationship of the Deadlock and Unsafe spaces



(b) The topology of the newly created partial deadlock

Figure 1: Illustration of the proof of Claim 1

by assumption, $\mathcal{H}$ did not contain any subsets satisfying the condition of Definition 4, prior to transition $s^i \rightarrow s^j$.

(iv) Since $C_k \geq 2$, in state s^i, R_k has a non-empty set of processes J^k allocated to it.

(v) Now consider the *resource dependency set* $\mathcal{DR}^*$ of processes in J^k at state s^j. This dependency set is defined as the union of the set of resources, say $\mathcal{R}_1$, being requested by the processes in J^k, with the set of resources, say $\mathcal{R}_2$, being requested by processes of resources in $\mathcal{R}_1$, with the set of resources, say $\mathcal{R}_3$, being requested by processes of resources in $\mathcal{R}_2$, etc. It is claimed that

a. $\mathcal{DR}^* \subseteq \mathcal{H} \cup \{R_k\}$, and

b. in particular, $R_k \in \mathcal{DR}^*$.

Claim (a) is true since otherwise $\mathcal{DR}^*$ contains a resource R_m which is not filled to capacity, and granting of any pending requests for resource R_k in state s^i produces no deadlock. But this violates our initial assumption about s^i being exactly one step away from deadlock.

Claim (b) is true since otherwise $\mathcal{DR}^* \subseteq \mathcal{H}$, and since $\mathcal{DR}^*$ represents a deadlock, our initial assumption that state s^i is deadlock-free is violated.

(vi) $R_k \in \mathcal{DR}^*$ implies that there exists a resource $R_p \in \mathcal{DR}^*$ with a process j_q requesting the free unit of capacity of R_k. Granting this request in s^i adds R_k to $\mathcal{H}$, but R_p, which belongs to $\mathcal{DR}^*$, can no longer belong to $\mathcal{H}$. But then, R_k cannot be involved in a deadlock since the dependency set $\mathcal{DR}^*$ of processes executing on R_k contains a resource, i.e., R_p, which has a unit of free capacity. Therefore, $\mathcal{H}$ does not contain a deadlocked subset, and the state s^k constructed by granting the request of process j_q is deadlock-free.

(vii) Summarizing the previous steps, we initially assumed the existence of unsafe states that are not partial deadlocks. This further implies the existence of a state s^i which is exactly one step away from a deadlock – i.e., granting any pending request in s^i leads to a deadlock. We then established that the existence of such a state implies

a. a non-empty set of resources $\mathcal{H}$ filled to capacity but not deadlocked,

b. a non-empty set of resources $\mathcal{G}$ with one unit of spare capacity, while

c. all requests that can be granted immediately are directed to $\mathcal{G}$ and no other resources. Further, the granting of any such request was to result in a deadlock.

However, when we took a member, R_k, of $\mathcal{G}$ for which there was a pending request for capacity in state s^i, it was found that

a. the non-unit capacity of the resources introduced us a non-empty set of processes already resident in that resource,

b. the fact that s^i was one transition away from deadlock required that the resource dependency set, $\mathcal{DR}^*$, of the abovementioned set of processes was precluded from containing resources other that R_k and members of $\mathcal{H}$,

c. the fact that s^i was not already a deadlocked state required that $\mathcal{DR}^*$ contain R_k, and finally

d. $\mathcal{DR}^*$ containing R_k implied the existence of a transition from s^i which produced another state s^k not in deadlock, a contradiction to the premise that s^i is a deadlock-free unsafe state exactly one step away from deadlock.

This last contradiction negates the original assumption of the existence of unsafe states which are not partial deadlocks, and establishes the truth of Claim 1. $\triangle$

The significance of this result, in the context of deadlock avoidance for Single-Unit RAS, stems from the fact that *detection* of (partial) deadlock in these systems is of polynomial complexity; an algorithm for this problem, with complexity $O(m^2\bar{C})$,[6] is given in the Appendix. The tractability of deadlock detection, combined with the established (above) nonexistence of unsafe states which are not already (partially) deadlocked, implies that for single-unit RAS in which every resource has non-unit capacity, the *optimal* deadlock avoidance policy can be *polynomially* obtained by *one-step look-ahead* on the state resulting by the satisfaction of a pending request: *if the resulting state is not a deadlock, then granting the request is a safe step.* In fact, this result seems to provide theoretical justification for a claim made by a number of researchers in the field (e.g., [24, 25]), that in the majority of the cases, deadlock avoidance policies based on a finite horizon of look-ahead steps, even though not provably correct, are well behaved.

In the discrete-part manufacturing context, the above result obtains an even more practical interpretation. Remember that according to the discussion accompanying the presentation of the RAS taxonomy in the introductory section, Single-Unit RAS models apply to FMS's in which jobs compete only for one unit of buffering capacity of the system component (i.e., workstation or material handling system) supporting their current processing requirements. Since emphasis is placed only on the *buffering* capacity of the system components, the condition

[6] where m denotes the number of resource types in the RAS, and $\bar{C}$ the maximal resource capacity among the system resources

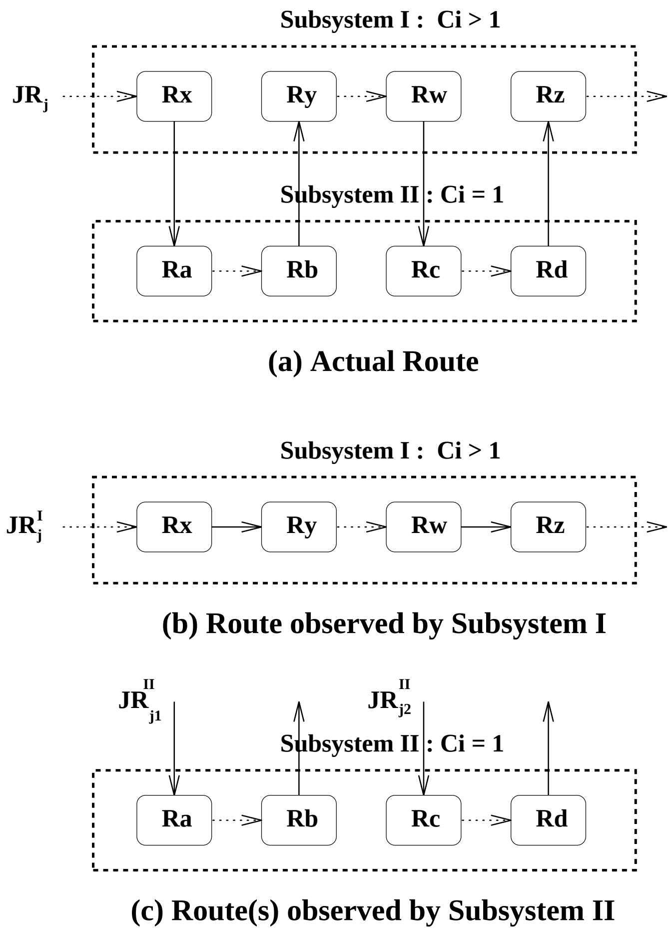


Figure 2: The proposed system decomposition

requested by Claim 1 for polynomially computable optimal deadlock avoidance, is immediately implementable in these environments: All that is required is to ensure that every system component is capable of simultaneously accommodating at least two jobs.

For the general case where a Single-Unit RAS consists of both unit and non-unit capacity resources, we propose a control policy based on the decomposition scheme indicated in Figure 2. The underlying idea is to manage the allocation of resources of non-unit capacity according to the optimal one-step look-ahead policy outlined above, while for the remaining part of the system consisting of unit capacity resources, for which no polynomially computable optimal control scheme is available, one of the developed Ω DAP's (e.g., [21, 23]) should be applied. Furthermore, since, in the general case, process routes will need the support of both resource types, the control of the two subsystems must be appropriately coordinated. The detailed specification of the proposed control scheme is as follows:

a. Divide the original RAS into two subsystems: (i) subsystem I, containing all the resources of non-unit capacity, and (ii) subsystem II, consisting of all unit-capacity resources.

b. Develop a separate model for the part-flow for each of these subsystems. Specifically, the part-flow model for subsystem I is obtained from the part-flow model of the original system by deleting from its process routes all stages that have to be executed on a unit-capacity resource. The part-flow model for subsystem II considers as separate process routes all the route segments of the original routes that have to be executed on unit-capacity resources.

c. Apply the optimal deadlock avoidance policy, implemented by the one-step look-ahead scheme described above, for the control of subsystem I, and any scalable and provably correct DAP (e.g., [3, 21, 23]) for the control of subsystem II. In addition, the following rules apply: (i) A process requesting a free resource unit of subsystem II, while allocated to a resource unit in subsystem I, can be granted its request only if the resulting state is admissible by both control policies. Furthermore, if the request is granted, the state in both models is immediately updated according to the corresponding process route observed by each subsystem. (ii) The same rule applies to the loading of a new process in the system through a resource of subsystem II. (iii) On the other hand, a process loaded into the system through a subsystem I resource, must be admitted only by the controller of that subsystem.

It should be clear from the description above, that the resulting policy is scalable and correct. Specifically, the correctness of the operation of subsystem II is established by the fact that this subsystem meets exactly the specifications used in the development of the applied DAP's, while the correctness of the operation of subsystem I is due to the fact that a process executing a route segment in subsystem II, has a unit of capacity reserved on the resource through which it re-enters subsystem I, and its occupation of that unit has already been pre-approved by the optimal policy for subsystem I. Since subsystem I is operated optimally (i.e., with maximal permissiveness), it is expected that such a policy would function efficiently in systems with most resources presenting a capacity equal to or greater than two (2), and with process routes making short excursions into resources of unit capacity.

Example: As an example, consider the system presented in Figure 3, which is a typical layout for many contemporary automated manufacturing cells. It consists of a number of workstations served by a robotic manipulator. Each workstation possesses an input and an output buffer, while the manipulator can carry only one part at a time. Hence, the presented system is a natural candidate for the application of the decomposition scheme discussed above.

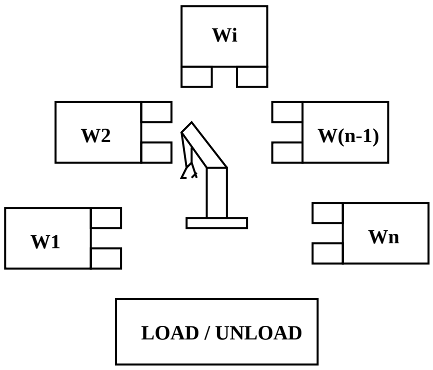


Figure 3: Example: The Automated Manufacturing Cell

An interesting remark that is pertinent for the system of Figure 3, is that the job excursions in subsystem II (i.e., the robotic manipulator) are of one-step length. As a result, the *hold-and-wait* condition, which is necessary for deadlock formation [11], does not apply to this (sub-)system, and therefore, its operation is deadlock-free. Hence, in this particular application, by focusing on subsystem I and controlling the buffering capacity of the system workstations through the optimal DAP, we control the entire system optimally, even though it contains unit-capacity resources. In fact, it is not difficult to see that the above scheme of controlling a RAS by one-step look-ahead on the resource allocation of subsystem I provides a correct DAP for the entire subclass of Single-Unit RAS in which jobs take only single-step excursions on subsystem II (i.e., the unit-capacity resources). Notice, however, that in the general case, this scheme is not necessarily optimal. $\triangle$

4 Correct and Scalable DAP's for Conjunctive RAS

4.1 The DAP Class Ω

Since we have shown how the aforestated optimality result can conclusively resolve the deadlock avoidance problem in discrete-part manufacturing systems modeled according to the Single-Unit RAS, in this section we address the problem of developing correct and scalable DAP's for the more complicated Conjunctive RAS model. To this end, we extend and generalize our past work, developed for the Single-Unit RAS case. Specifically, in an effort to design *correct* (i.e., provably deadlock and restricted-deadlock free) and *scalable* (i.e., of polynomial complexity in setup and real-time execution) DAP's despite the difficulties arising from the increased (nonpolynomial) complexity of the state-safety problem, a certain class of policies, characterized as class Ω

DAP's, were proposed as good candidates in [14, 21]. In particular, DAP class Ω was defined to be the set of policies that can be formulated as *a set of "static"[7] inequalities which constrain the allocation of system resources at any single instance*. The policy constraints are selected so that (i) their evaluation for a system state is of polynomial computational complexity (and therefore implementable in real-time), and (ii) they guarantee deadlock and restricted deadlock-free operation of the system. A feasible event is enabled under such a policy, iff it does not violate the policy constraints. This idea is extensible to the deadlock avoidance framework pertaining to the Conjunctive RAS model, which was developed in Section 2. Furthermore, the Integer Programming (IP) characterization of state safety presented in the last part of Section 2.1, provides the basis for the formal analysis of the policy correctness. Specifically, similar to the Single-Unit RAS case, the following proposition holds true:

Proposition 1 *Let $\mathcal{P} \in \Omega$. If the existence of a set of variables $\{\delta_{ik\tau}\}$ satisfying the "state-safety" Equations 2, 3, 7 and the policy constraints for step $t = \tau$ implies the existence of a set of variables $\{\delta_{ik(\tau+1)}\}$ satisfying the same set of equations for step $t = \tau + 1$, and the combined set of variables $\{\delta_{ik\tau}, \delta_{ik(\tau+1)}\}$ satisfies the "state-safety" Equations 5 and 6 (possibly together with some additional policy "linking" constraints), policy $\mathcal{P}$ is* correct.

The formal proof of Proposition 1 presents an identical structure to the proof of its counterpart for Single-Unit RAS, presented in [21], and therefore, it is omitted. It is, however, emphasized that the meaning of the indicator variables $\{\delta_{ik\tau}\}$ as well as the detailed equation content in each of the two RAS models are different (c.f., relevant discussion in the introductory section).

The analysis of a policy $\mathcal{P} \in \Omega$ in the context of the framework established by Proposition 1 simultaneously resolves the problem of policy correctness w.r.t., both, deadlocks and restricted deadlocks. In [21], this framework was used for establishing the correctness of the *RUN (Resource Upstream Neighborhood)* Ω DAP, and for providing an alternative proof for the correctness of the *Deadlock Avoidance Algorithm (BKDAA)*, initially developed in [3]. Both of these policies in their current formulation are applicable to the case of single-unit RAS. In the following, RUN is generalized so that it applies to the broader AND model. Furthermore, another policy developed in our research program for Single-Unit RAS, the *Resource Ordering (RO)* policy [23], is extended to apply to the class of Single-Type RAS. For each policy, the proposed generalization is motivated by a series of observations on the underlying mechanism

[7] By this it is implied that the constraints consider only the present status of the resource allocation and not its history.

that establishes the policy functionality in the single-unit RAS case, while formal proofs of their correctness are provided in the Appendix.

4.2 Generalizing RUN for Conjunctive RAS

In the single-unit RAS case, RUN was motivated by the remark that if there are some resources with higher capacity than others in the system, then they can function as temporary buffers for the processes that they support [21]. Hence, every resource R_i is associated with a set of process stages in the sense that if a process is currently executing one of these stages then it must have also one unit of capacity reserved on resource R_i. The set of job stages thus associated with resource R_i define its *neighborhood* $\mathcal{N}_i$. In the Conjunctive RAS model, it is not possible to map a process stage to a single supporting resource, and therefore, this line of reasoning is not functional. However, it is still possible to generalize RUN to the Conjunctive RAS model, if one takes a closer look at the proof of the correctness of RUN for the Single-Unit RAS, and attempts to analyze the mechanism which establishes the policy correctness in that case [21]. Viewed from this perspective, RUN can be considered as a *partial resource reservation system*, with the reservations decomposed over segments of the process routes. Furthermore, the detailed neighborhood construction is based on an *ordering* imposed on the system resources, which guarantees that there is always a resource in the system – specifically, one of *minimal* order among those containing processes – the processes of which can advance to their nest stage. It is this operational scheme that the policy generalization seeks to establish for the Conjunctive RAS case. The details of the policy definition are given below. Notice that contrary to the Single-Unit RAS case, in this (re-)definition of the policy, neighborhoods are attributes of the distinct process stages, instead of the system resources.

RUN re-definition for the Conjunctive RAS model:

a. Impose an (partial) ordering on the resource set, $o : \mathcal{R} \rightarrow |\mathcal{R}|$, and let $R_i \succeq (\preceq) R_j$ imply that $o(R_i) \geq (\leq) o(R_j)$. The ordering induced by the resource capacities seems to be a good (natural) choice.

b. Define the *reservation span* for route stage JT_{ik} to be the subsequence $< JT_{i,k-n}, \ldots, JT_{ik} >$ s.t. $JT_{i,k-n-1}$ is the first stage preceding JT_{ik} that satisfies the condition:

$$\min\{o(R_q) : \; JT_{i,k-n-1}[R_q] > 0\} > \max \bigcup_{j=0}^{n} \{o(R_q) : \; JT_{i,k-j}[R_q] > 0\} \qquad (8)$$

In other words, the reservation span of stage JT_{ik} ends at a route stage which requires resources not involved in the processing of the stages covered by the reservation span, and in this way, the needs of the route segment covered by the reservation span are completely decomposed from the needs of the preceding stage.

c. Having computed the reservation span of all route segments, define the *neighborhood* of stage $JT_{ik},\ i = 1,\ldots,n,\ k = 1,\ldots,l(i)$:

$$\mathcal{N}_{ik} = \{JT_{il} :\ JT_{ik} \text{ is in the reservation span of } JT_{il}\} \tag{9}$$

d. Redefine the *nominal* requirements of stage $JT_{ik},\ i = 1,\ldots,n,\ k = 1,\ldots,l(i)$, to be the resource allocation vector that can meet the resource needs of the entire neighborhood $\mathcal{N}_{ik}$, i.e.,

$$JT_{ik}^{\mathcal{P}}[q] = \max\{JT_{il}[q] :\ JT_{il} \in \mathcal{N}_{ik}\}, \quad \forall q \tag{10}$$

e. Then, a formal statement of RUN DAP for the Conjunctive RAS model is that *a process can proceed to the execution of its next processing stage if and only if it can be allocated the* nominal *requirements of this stage defined in Step 4 above.*

The following example will help clarify the policy definition.

Example: Consider the RAS system with resource set $\mathcal{R} = \{R_1, R_2, R_3, R_4, R_5\}$ running a single process route

$$JT =< \begin{bmatrix} a_{11} \\ a_{12} \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ a_{23} \\ 0 \\ a_{25} \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ a_{34} \\ 0 \end{bmatrix}, \begin{bmatrix} a_{41} \\ 0 \\ a_{43} \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ a_{53} \\ 0 \\ a_{55} \end{bmatrix}, \begin{bmatrix} 0 \\ a_{62} \\ 0 \\ 0 \\ 0 \end{bmatrix} >$$

where a_{ij} denotes the requirements in resource R_j of stage JT_i.

Using the natural ordering of the resource set in the policy implementation, i.e., $o(R_i) = i,\ i = 1,\ldots,5$, the reservation span of JT_6 is just stage JT_6, since the minimally ordered resource required for the execution of stage JT_5 is R_3, which is higher in the ordering than R_2, the maximally ordered resource required for the execution of JT_6. However, Equation 8 is not satisfied at any point moving backwards while trying to establish the reservation span of stage JT_5, and thus, this reservation span consists of all stages JT_1 through JT_5. The complete set of reservation spans, neighborhoods, and nominal requirements for all stages in this example, are tabulated in Table 1. It can be seen that the original stage requirement vectors have been augmented to cover the needs of the entire neighborhood associated with their stages. $\triangle$

stage	reservation span	neighborhood	nominal requirements
1	$< JT_1 >$	$\{JT_1, JT_2, JT_3, JT_5\}$	$< a_{11}, a_{12}, \max\{a_{23}, a_{53}\}, a_{34}, \max\{a_{25}, a_{55}\} >$
2	$< JT_1, JT_2 >$	$\{JT_2, JT_3, JT_5\}$	$< 0, 0, \max\{a_{23}, a_{53}\}, a_{34}, \max\{a_{25}, a_{55}\} >$
3	$< JT_1, JT_2, JT_3 >$	$\{JT_3, JT_5\}$	$< 0, 0, a_{53}, a_{34}, a_{55} >$
4	$< JT_4 >$	$\{JT_4, JT_5\}$	$< a_{41}, 0, \max\{a_{43}, a_{53}\}, 0, a_{55} >$
5	$< JT_1, \ldots, JT_5 >$	$\{JT_5\}$	$< 0, 0, a_{53}, 0, a_{55} >$
6	$< JT_6 >$	$\{JT_6\}$	$< 0, a_{62}, 0, 0, 0 >$

Table 1: Example: Generalizing RUN for Conjunctive RAS

The following Proposition, formally proven in the Appendix, establishes the correctness of the above policy.

Proposition 2 *The above re-definition of RUN for the Conjunctive RAS model is a* correct *DAP.*

Furthermore, it can be easily seen from the policy definition that computing the stage nominal requirements is of no higher complexity than $O(mnL^2)$, where m is the number of resource types, n is the number of process types, and L is the maximal route length. As it is also the case for the Single-Unit RAS [21], taking the *disjunction* of a polynomially bounded number of policy realizations obtained by imposing different orderings on the system resources, results in a correct scalable DAP, which is potentially more powerful than its constituent parts. Finally, it is easy to show that when every stage consists of a single resource unit requirement, the above presented definition of RUN simplifies to the policy definition presented in [21].

4.3 Generalizing RO for Single-Type RAS

In the following discussion we shall focus on a specialization of the general conjunctive RAS model. This specialization is characterized by the fact that every stage of the allowed process types requires only a single resource type for its execution. Contrary, though, to the single-unit model, it is possible for a process to require more than one units of a single resource at a stage. For this reason, the considered RAS class will be called the *single-type* RAS model. As it was mentioned in the Introduction, the model is applicable to the operation of discrete-part manufacturing systems in which jobs are processed into batches of varying sizes.

Since this class is a subcase of the general conjunctive model, it is obvious that the RUN policy defined in the previous section, applies in this case, too. The purpose of this section, however, is to show that the *RO* policy, initially presented in [23] for the single-unit RAS model, can be generalized to cover the single-type RAS model, as well. Our interest in generalizing RO for this case stems from our experimental experience with the policy performance in the single-unit case, where it seems that RO allows a considerably higher level of flexibility to the system than the corresponding RUN implementations (cf. [14]). Similar to the RUN case, in the sequel we present the motivation behind the policy generalization, while a formal proof for its correctness is provided in the Appendix.

RO for the single-unit RAS case, originated from the observation that in a "counterflow" system – i.e., a RAS in which it is possible to number the system resources so that process routes correspond to strictly increasing or strictly decreasing resource sequences –, deadlock-free operation can be ensured as long as no pair of resources (R_i, R_j) with $i < j$ are filled to capacity with the processes in resource R_i corresponding to ascending resource sequences, and the processes in resource R_j corresponding to descending resource sequences. In the single-type RAS case, viewing the process routes as "resource strings" is still valid. However, the increased requirements in resource units posed by a process stage require the modification of the condition that guarantees that for every pair of resources there exists an "open passage" either to the left or to the right. This leads to the following policy redefinition for Single-Type RAS.

RO re-definition for the Single-Type RAS model: For the case of Single-Type *counterflow* systems, a natural restatement of the policy defining condition is as follows: Define as *left (right) - directed* processes occupying resource R_i, those with decreasing (increasing) route sequences w.r.t. the resource ordering, and let LC_i (RC_i) denote the number of R_i units occupied by left (right) - directed processes. Furthermore, let l_i (r_i) denote the maximal resource requirements among the left (right) - directed stages supported by resource R_i. Considering any pair of resources R_i, R_j with $R_i < R_j$, there cannot be a deadlock among the processes they support as long as $l_i \leq C_i - RC_i$ or $r_j \leq C_j - LC_j$, i.e.,

$$\forall i,j, \quad R_i < R_j \Longrightarrow (RC_i \leq C_i - l_i) \vee (LC_j \leq C_j - r_j) \tag{11}$$

Notice that for the single-unit counterflow RAS, $l_i = r_i = 1, \forall i$, and Equation 11 reduces to

$$\forall i,j, \quad R_i < R_j \Longrightarrow RC_i + LC_j \leq C_i + C_j - 1 \tag{12}$$

which is the original policy-defining equation for Single-Unit RAS [23].

Extending the policy to noncounterflow single-type RAS requires some modifications so that the policy compensates for the behavior of the *undirected* process segments, i.e., the process subsequences that do not present any monotonicity w.r.t. the imposed resource ordering.

However, these modifications are more complicated than in the single-unit RAS case, and can summarized as follows:

a. Instead of the parameters l_i and r_i associated with resource R_i in the counterflow case, define the single parameter d_i to be equal to the maximal resource requirements presented by any stage supported by resource R_i.

b. Furthermore, define the *nominal* requirements of any undirected step supported by resource R_i to be equal to d_i.

c. Include in the definition of, both, LC_i and RC_i, the occupancy of resource R_i due to the undirected process instances, as it is expressed by their nominal requirements.

d. Then, the policy is defined by the condition:

$$\forall i, j, \quad R_i < R_j \Longrightarrow (RC_i \leq C_i - d_i) \vee (LC_j \leq C_j - d_j) \tag{13}$$

Example: The following example will elucidate the previous definitions. Consider the single-type RAS with resource set $\mathcal{R} = \{R_1, R_2, R_3, R_4\}$ and process routes:

$$\begin{array}{lllll} JT_1: & R_1 & R_2 & R_3 & R_4 \\ JT_2: & R_4 & R_3 & R_1 & \\ JT_3: & R_1 & R_3 & R_2 & R_4 \end{array}$$

Notice that w.r.t. the natural resource ordering, routes JT_1 and JT_2 are right and left-directed, respectively, while route JT_3 is undirected, with subsequence $< JT_{31}, JT_{32} >$ being its undirected segment. The LC_i and RC_i, $i = 1, \ldots, 4$, contents and the definition of the d_i, $i = 1, \ldots, 4$, parameters is given in Table 2. In this table, a_{ij} denotes the requirements of stage JT_{ij}, in units of its supporting resource.

resource	LC_i	RC_i	d_i
R_1	$\#(JT_{31})$	$\#(JT_{11})$, $\#(JT_{31})$	$\max\{a_{11}, a_{31}\}$
R_2	–	$\#(JT_{12})$, $\#(JT_{33})$	$\max\{a_{12}, a_{33}\}$
R_3	$\#(JT_{22})$, $\#(JT_{32})$	$\#(JT_{13})$, $\#(JT_{32})$	$\max\{a_{13}, a_{22}, a_{32}\}$
R_4	$\#(JT_{21})$	–	a_{21}

Table 2: Example: Generalizing RO for single-type RAS

Furthermore, the nominal requirements of the undirected stages JT_{31} and JT_{32} are $d_1 = \max\{a_{11}, a_{31}\}$ and $d_3 = \max\{a_{13}, a_{22}, a_{32}\}$, respectively. Finally, notice that since a process instance executing its last stage in the system can be unloaded before any other steps take place, the last stages of the process routes are ignored in the policy-defining constraints. $\triangle$

Proposition 3 *The generalization of RO for single-type RAS, described by steps 1 – 4 above, is a* correct *policy.*

A formal proof for Proposition 3 is provided in the Appendix. The initial setup of the policy is of complexity $O(D)$, where D is the total number of distinct route stages in the system, while the number of constraints that have to be checked in real-time operation is $O(m^2)$, where m is the number of system resources. The policy parameterization w.r.t. the resource ordering implies that the above policy definition actually generates a *family* of DAP's for these system, each member of which is obtained by implementing the policy w.r.t. a certain resource permutation. Furthermore, the policy resulting from the *disjunction* of any set of policies from this family is another correct DAP; the policy is also scalable as long as the cardinality of the policy-defining set is kept polynomially bounded w.r.t. the system size (i.e., the number of resource types and the number of distinct route stages).

5 Conclusions

In the first part of this paper, we established the important result that for single-unit RAS with non-unit resource capacities, there are no deadlock-free unsafe states in the system operation. The practical implication of this result is that the optimal deadlock-avoidance policy for these systems is polynomially obtainable (tractable) in real-time through one-step look-ahead. Furthermore, a new scalable and correct control scheme, employing a system decomposition based on the resource capacities, was developed for the more general class of single-unit RAS. An arising question is whether the obtained result on the nonexistence of deadlock-free unsafe states can be extended to other RAS subclasses; this is part of our current research agenda.

In the second part of the paper, RUN and RO DAP's – two correct and scalable DAP's initially developed for single-unit RAS – were generalized so that they apply to more general RAS classes. It was also observed that both DAP's actually define entire families of policies for a given RAS, the members of which result from the different orderings imposed on the RAS resource set during the DAP implementation. An interesting problem is the determination of those orderings that lead to the least restrictive policy implementation. For a more detailed definition of the problem of "optimal orderings" in the context of single-unit RAS, and some approaches to its solution, the reader is referred to [14, 19, 23]. It is believed that the basic ideas and remarks expressed in these documents carry over to the conjunctive RAS case.

Finally, starting from the observation that the operation of the system resources under any of the proposed deadlock avoidance schemes becomes considerably coupled, a naturally arising issue is the development of performance-oriented scheduling policies that are appropriate for this environment. Some initial steps towards this direction can be found in [18, 19, 20, 22].

A Appendix

A.1 A Deadlock-Detection Algorithm (DDA) for the Single-Unit RAS Model

We present here an algorithm which detects whether a given single-unit RAS state s is a deadlock or not.

To facilitate the further discussion, let us define the *Resource Dependency Vector (RDV)* with one component for every RAS resource type R_i, containing the following information:

- the number of R_i units allocated to processes in state s,
- a list of the resource types required by the processes currently allocated to R_i, for their next step.

This information can be easily obtained from the state vector and the process routes.

The algorithm considers the RDV and tries to extract a subset of resources which satisfies Definition 4 of Section 3. Specifically, it goes through a number of scannings of RDV, and at every scanning it eliminates a number of resources from further consideration. The resources eliminated are those which either (i) are not filled to capacity or (ii) have units allocated to processes which are ready to leave the system or to move to an already eliminated resource. Proceeding in this way, the algorithm either eliminates all resources, in which case state s is safe, or at a certain scanning no resource is eliminated, in which case state s is a deadlock.

We remark here that this algorithm is a modification of the idea of *cycle detection in a digraph through topological ordering ([1])*, in a way that it applies to the considered problem.

A detailed statement of the algorithm has as follows:

algorithm DDA
begin
  /* Initialize */

```
    CANDIDATES := RDV;
    STUCK := FALSE;

    /* processing step */
    while (not-empty(CANDIDATES) and not(STUCK)) do
    begin
       STUCK := TRUE;
       for (i:=1 to cardinality(CANDIDATES)) do
       begin
          if (deleteable(R_i))
          begin
             eliminate(R_i);
             STUCK := FALSE;
          end
       endfor
    endwhile

    if (not-empty(CANDIDATES))
       state s is a deadlock with resources in CANDIDATES being the deadlocked subset;
    else
       state s is deadlock-free;
end
```

In the previous description, CANDIDATES is a list holding the resources which have not been eliminated yet, and it is possible that they contain a deadlock. Function deleteable(R_i) evaluates to TRUE if resource R_i has either idle units of capacity or it is occupied by some processes which are not waiting for resources in the CANDIDATES list. Procedure eliminate(R_i) removes R_i from the CANDIDATES list.

Finally, we have the following result about the complexity of this algorithm:

Proposition 4 *The complexity of the DDA algorithm is $O(m^2 \cdot \bar{C})$, where m is the number of the RAS resource types, and $\bar{C} = \max_{i \in \{1,\ldots,m\}} C_i$ (i.e., the maximum resource capacity).*

Proof: For every resource, function deleteable() has to examine upto $\bar{C}$ resource units of capacity. Furthermore, at every outer cycle, deleteable is called as many times as the current size of the CANDIDATES list. However, notice that this size is reduced by at least one at every iteration, since at least one resource has to be eliminated from the `CANDIDATES` list for the algorithm to continue. Thus, in the worst case, the number of single resource capacity units to be worked upon, is:

$$m\bar{C} + (m-1)\bar{C} + \ldots + 1 \cdot \bar{C} = \frac{m(m+1)}{2}\bar{C}$$

which is $O(m^2 \cdot \bar{C})$. $\triangle$

A.2 Proof of Proposition 2

Starting from the result stated in Proposition 1, all we have to show in order to prove the correctness of RUN, is that for every state accepted by the policy (i.e., meeting the policy-defined nominal requirements), there exists at least one process able to advance to its next stage, without violating the policy. We discern three basic cases.

Case 1: There is a process ready to leave the system. Obviously, unloading the finished process trivially satisfies the proof claim.

Case 2: There exists a process advancing to a stage which is in the neighborhood of its current stage. By the policy definition, if process stage $JT_{i,j+1}$ belongs to the neighborhood $\mathcal{N}_{ij}$ of stage JT_{ij}, then, $\mathcal{N}_{i,j+1} \subset \mathcal{N}_{ij}$. This further implies that the nominal requirements of JT_{ij} is a superset of those for $JT_{i,j+1}$ (cf. Eq. 10). Hence, the considered process can proceed to its next stage.

Case 3: All processes are advancing to stages which do not belong to the neighborhoods of their current stages. Let resource R_{min} be a minimally ordered resource which has some of its units busy, and let process P_i be one of the processes which have been allocated units of R_{min}. Furthermore, let JT_{ij} be the stage currently executed by P_i. Since stage $JT_{i,j+1}$ is not in the neighborhood of stage JT_{ij}, and P_i still holds units of the minimally ordered busy resource, it follows, from the policy definition, that the resources required for the execution of stage $JT_{i,j+1}$ are currently idle. Hence, P_i's immediate requests can be satisfied. Notice, however, that the neighborhood of $JT_{i,j+1}$ might include some other successor stages, which might differentiate the nominal requirements from the actual requirements for stage $JT_{i,j+1}$. The following observations establish the feasibility of advancing P_i w.r.t. the process nominal requirements: (i) If the modifications differentiating the nominal from the immediate requirements for stage $JT_{i,j+1}$ concern resources lower than R_{min} in the ordering, the nominal requirement can still be met due to the current idleness of these resources. (ii) If, on the other hand, the modifications concern resources among those currently busy, the policy definition guarantees that the modification-inducing stages are also in the neighborhood of stage JT_{ij} (because stage JT_{ij} is immediately preceding stage $JT_{i,j+1}$, and resource R_{min} is a minimally ordered busy resource) and thus, the new nominal requirements for these resources are covered by the corresponding nominal requirements of stage JT_{ij}.

Since cases $1 - 3$ exhaustively cover all possible RAS states, the proof is complete. $\triangle$

A.3 Proof of Proposition 3

The proof of Proposition 3 is also aligned to the framework established by Proposition 1, i.e., all we have to show is that for every RAS state accepted by the policy, there exists at least one process instance able to proceed to its next stage. To show this, we discern a number of cases:

Case I: Suppose that there exist a number of resources R_i s.t. $RC_i > C_i - d_i$, and let R_k be the rightmost among them. Then,

a. *if R_k contains a right-directed process j_k heading to resource R_l with sufficient free capacity*, process j_k can proceed, since this movement increases only RC_l, and for any resource $R_x > R_l > R_k$, it holds that $d_x \leq C_x - LC_x$.

b. *if R_k contains right-directed processes but all these processes are heading to resources with insufficient slack*, consider any of them, let's say process j_k heading to resource R_l. Since $\texttt{slack}(R_l) < \texttt{req}(j_k) \leq d_l$, it follows that $\texttt{slack}(R_l) \neq C_l - LC_l$ (since, by the policy definition, it has to be $C_l - LC_l \geq d_l$), and thus, R_l contains at least one right-directed process, j_l. Now, if j_l is heading to R_t with sufficient slack, j_k can proceed, according to the argument of subcase (I.1). O.w., applying the reasoning outlined above, it follows that R_t contains a right-directed process j_t, and we can repeat the entire argument focusing on j_t. Since the resource set $\mathcal{R}$ is finite and ordered, recursive implementation of this argument implies that eventually we shall identify a process able to proceed.

c. *if R_k contains only undirected processes*, w.l.o.g., suppose that it contains a process j_k requiring an immediate step to the right, to resource R_l. If R_l does not have sufficient slack capacity, the reasoning of subcase (I.2) applies. If R_l does have sufficient slack, we shall show that allowing j_k to advance does not violate the policy. To this end, notice that: (a) Satisfaction of the policy constraints concerning RC_l and LC_x for $R_x > R_l$ is resolved by applying the reasoning of subcase (I.1). (b) Satisfaction of the policy constraints concerning LC_l and RC_x for $R_k < R_x < R_l$, is guaranteed by the assumption that R_k is the rightmost resource with $RC_k > C_k - d_k$. (c) Satisfaction of the policy constraint concerning LC_l and RC_k is guaranteed by the fact that, since j_k is an undirected process, its advancement reduces RC_k by d_k. (d) Satisfaction of the policy constraints concerning LC_l and RC_x for $R_x < R_k$, is guaranteed by the fact that, since all processes in R_k are undirected, it also holds that $LC_k > C_k - d_k$, which further implies that $RC_x \leq C_x - d_x$, $\forall R_x < R_k$.

This resolves case (I).

Case II: Suppose that there exist a number of resources R_i *s.t.* $LC_i > C_i - d_i$, *and let* R_k *be the leftmost among them.* Then, we can apply a line of reasoning which is dual to that applied in case (I), in the sense that (a) we are dealing with motion in the left direction, (b) we change RC to LC and vice versa, (c) we reverse the inequalities in resource orderings, and (d) we change "rightmost" with "leftmost".

Case III: For all resources R_i, $RC_i \leq C_i - d_i$ *and* $LC_i \leq C_i - d_i$. First notice that if a process j_i waits upon a resource R_w with free capacity, then j_i can proceed, since $\forall\ R_x < R_w$, $RC_x < C_x - d_x$, and $\forall\ R_x > R_w$, $LC_x < C_x - d_x$. It remains to be shown that such a process will always exist. Suppose not, and consider the leftmost resource requested by any of these stuck processes, denoted by R_{min}. Then, R_{min} has $\texttt{slack}(R_{min}) < d_{min}$, since D_{min} is the maximal request that can be posted for resource R_{min}. Furthermore, it follows from the defining characterization of R_{min}, that all processes currently allocated to it, request an immediate step to the right. Therefore, $RC_{min} = C_{min} - \texttt{slack}(R_{min})$ which further implies that $C_{min} - RC_{min} < d_{min}$ and contradicts the case assumption.

Cases (I) - (III) cover exhaustively all possible RAS states, and thus, the proof is complete. $\triangle$

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] T. Araki, Y. Sugiyama, and T. Kasami. Complexity of the deadlock avoidance problem. In *2nd IBM Symp. on Mathematical Foundations of Computer Science*, pages 229–257. IBM, 1977.

[3] 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.

[4] J. Brzezinski, J-M. Helary, M. Raynal, and M. Singhal. Deadlock models and general algorithms for distributed deadlock detection. Technical report, The Ohio State University, 1994.

[5] E. G. Coffman, M. J. Elphick, and A. Shoshani. System deadlocks. *Computing Surveys*, 3:67–78, 1971.

[6] E. W. Dijkstra. Cooperating sequential processes. Technical report, Technological University, Eindhoven, Netherlands, 1965.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., New York, NY, 1979.

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

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

[10] J. W. Havender. Avoiding deadlock in multi-tasking systems. *IBM Systems Journal*, 2:74–84, 1968.

[11] R. D. Holt. Some deadlock properties of computer systems. *ACM Computing Surveys*, 4:179–196, 1972.

[12] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, MA, 1979.

[13] M. Lawley, S. Reveliotis, and P. Ferreira. Design guidelines for deadlock handling strategies in flexible manufacturing systems. *Intl. Jrnl. of Flexible Manufacturing Systems (to appear)*, pages –, 1996.

[14] M. A. Lawley. *Structural Analysis & Control of Flexible Manufacturing Systems*. PhD thesis, University of Illinois, Urbana, IL, 1995.

[15] Y. T. Leung and G. J. Sheen. Resolving deadlocks in flexible manufacturing cells. *Journal of Manufacturing Systems*, 12:291–304, 1994.

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

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

[18] S. A. Reveliotis. "performance lp's" for multi-server queueuing networks with blocking and non-exponential service times. Technical Report UIUL-ENG 96-4013 (submitted to the IEEE Trans. on Automatic Control), Dept. of Mechanical & Industrial Eng., UIUC, 1996.

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

[20] S. A. Reveliotis and P. M. Ferreira. An analytical framework for evaluating and optimizing the performance of structurally controlled fms. In *1996 IEEE International Conference on Robotics and Automation*, pages 864–869. IEEE Robotics and Automation Society, 1996.

[21] S. A. Reveliotis and P. M. Ferreira. Deadlock avoidance policies for automated manufacturing cells. *IEEE Trans. on Robotics & Automation (to appear)*, 12:–, 1996.

[22] S. A. Reveliotis and P. M. Ferreira. Performance evaluation of structurally controlled fms: Exact and approximate approaches. In *Flexible Automation and Intelligent Manufacturing, 1996*, pages 829–838. Manufacturing Research Center, Georgia Tech., 1996.

[23] 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.

[24] 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.

[25] 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.