A HIERARCHICAL CONTROL SYSTEM FOR SCHEDULING AND SUPERVISING FLEXIBLE MANUFACTURING CELLS

by

SHERIF FAHMY

A thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY
in
MECHANICAL AND MANUFACTURING ENGINEERING

Department of Mechanical and Manufacturing Engineering
University of Manitoba
Winnipeg, Manitoba, Canada

Abstract

A hierarchical control system is proposed for automated flexible manufacturing cells (FMC) that operate in a job shop flow setting. The control system is made up of a higher level scheduler/reactive scheduler, which optimizes the production flow within the cell, and a lower level supervisor that implements the decisions of the scheduler on the shop floor. Previous studies have regularly considered the production scheduling and the supervisory control as two separate problems. This has led to: i) deadlock-prone optimized schedules that cannot be implemented in an automated setting, ii) deadlock-free optimized schedules that lack the means to be transformed into shop floor supervisors, or iii) supervisors that can safely drive the system with no consideration for production performance. The proposed control system combines mathematical models and an insertion heuristic to solve the deadlock-free scheduling problem in job shops, a deadlock-free reactive scheduling heuristic that can revise the schedules upon the occurrence of a wide variety of disruptions, and a systematic procedure that can transform schedules into readily implementable Petri net (PN) supervisors. The integration of these modules into one control hierarchy guarantees a correct, optimized and agile behavior of the controlled system.

The performances of the mathematical models, the scheduling and the reactive scheduling heuristics were evaluated by comparison to performances of previous approaches. Experimental results showed that the proposed modules performed consistently better than the other corresponding approaches. The supervisor realization procedure and the overall control architecture were validated by simulation and

implementation in an experimental robotic FMC. The control system developed was capable of driving the experimental cell to satisfactorily complete the processing of different product mixes that featured complex processing routes through the cell.

Acknowledgement

I wish to offer my sincerest thanks to my advisors, Dr. Subramaniam Balakrishnan and Dr. Tarek ElMekkawy, for their patience, guidance, friendship, support, and encouragement throughout this work. The opportunity that they have provided me to start this work has been a life changing experience. Special thanks also to Mr. Ken Tarte who provided all the necessary help during the final phase of the research in a timely and efficient manner. I would also like to thank Dr. Aniruddha Gole, Dr. Norman Richards, and Dr. Gary Wang, my advisory committee members, for their valuable and constructive suggestions and support throughout the work. My deepest gratitude for Dr. M. Adel Shalaby for all what I have learned from him during my early days as a researcher. Finally, I wish to thank the University of Manitoba for the financial assistance they provided me throughout these years.

Dedicated to my late Father, my caring Mother,
my precious and compassionate wife Amira,
and my princess and lovely daughter Nadin.

Table of Contents

List of Tables

List of Figures

Nomenclature

$\alpha_{(j\text{-}l)}$ Control action signal to transport a job from machine j to machine l

β_1 Control feedback condition to indicate availability of the robot

β_{2j} Control feedback condition to indicate acquisition of resource j

A Rank Matrix

AGV Automated Guided Vehicle

A_{ik} Set of alternative machines for operation o_{ik}

AMS Automated Manufacturing System

AOR Affected Operations Rescheduling

APN Automation Petri Net

ASMG Augmented Scheduling Marked Graph

ASRS Automated Storage and Retrieval Systems

B Capacity of central buffer

BF Best First Strategy

$b_{ik} = \begin{cases} 1 \text{ if job } i \text{ resides in a buffer before starting } o_{ik} \\ 0 \text{ otherwise,} \end{cases}$

$b_{ikpr1} = \begin{cases} 1 \text{ if } o_{ik} \text{ is started after } o_{p(r-1)} \text{ is completed} \\ 0 \text{ otherwise,} \end{cases}$

$b_{ikpr2} = \begin{cases} 1 \text{ if } o_{ik} \text{ is started before } o_{pr} \text{ is started} \\ 0 \text{ otherwise.} \end{cases}$

B_j Capacity of intermediate buffer before (input buffer to) machine j

BS Beam Search Strategy

BUFP Buffer presence factor

CB	Central buffer
CBA	MIP Model Featuring a Central Buffer with Arbitrary Capacity
CMS	Cellular Manufacturing System
CNC	Computer Numerical Control
CPN	Colored Petri Nets
CPPN	Controlled Production Petri Net
DEDS	Discrete Event Dynamic System
DES	Discrete Event System
DEV	Deviation from original schedule in the reactive schedule
DG	Disjunctive Graph
DMS	Dedicated Manufacturing System
DP	Dynamic Programming
E	Set of operations that can be eliminated
E	Small positive number
F	Firing Vector of a Petri net
FLX	System Flexibility factor
FMC	Flexible Manufacturing Cell
FMS	Flexible Manufacturing System
GA	Genetic algorithm
GDRS	Generic Deadlock-free Reactive Scheduling Tool
GT	Group Technology
I	Set of all Jobs $\{1, 2, ..., n\}$
I_i	Input buffer of job i

IB1	MIP Model Featuring Intermediate Buffers with Unit Capacity
IBA	MIP Model Featuring Intermediate Buffers with Arbitrary Capacity
IBS	MIP Model Featuring Intermediate Buffers for Swapping
IM	Incidence Matrix of a Petri net
I/O	Input/Output
J	Set of all machines {1, 2, …, *m*}
JI	Job Insertion
LLD	Logic Ladder Diagram
m	Number of machines in the system
M	Marking of Petri net
M	large positive number
M_o	Initial Marking of a Petri Net
MAG	Magnitude of Disruption Factor
mAOR	Modified Affected Operations Rescheduling
MFT	Mean Flow Time
MG	Marked Graph
MIP	Mixed Integer Programming
MS	Makespan
MS_{org}	Makespan of Original Schedule
MS_{rev}	Makespan of Revised Schedule
MTD	Reactive Scheduling Method Factor
n	Number of jobs in the system
NEWJ	Number of New Jobs Factor

NMFT	Normalized Mean Flow Time
O	Set of ordered pairs of operations corresponding to precedence relations in routes of jobs
OB	Output buffer
O_i	number of operations of job i
OI	Operations Insertion Algorithm
o_{ik}	Operation k of job i
OPTIM	Operation times of new jobs factor
PC	Personal Computer
P/D	Pick-up/Drop-off
p_{ij}	Position in rank matrix (or place in PN) associated with the processing of job i on machine j
P_j	Set of operations processed on machine j
PLC	Programmable Logic Controller
PN	Petri Net
PPN	Production Petri Net
Q	Set of ordered pairs of operations corresponding to the processing order on the machines defined by the schedule
R_i	Number of alternative routes of job i
$R(M_o)$	Reachability set of a Petri Net
RFLX	Routing Flexibility Factor
RG	Reachability Graph
RMFT	Relative Mean Flow Time

RNERV Relative Nervousness

RSOLT Relative Solution Time

RSR Right Shift Rescheduling

RW Ramadge and Wonham

S Set of all operations that have successors in their jobs' routes

SC Supervisory Control

SCT Supervisory Control Theory

SFC Sequential Function Chart

SIZ System size factor

SMG Scheduling Marked Graph

ST Solution Time

TD Transition Digraph

T_{ikj} Processing time of operation k of job i on machine j

$t_{j\text{-}s}$ Transition of PN associated with the release of machine j and acquisition of machine s

TOI Transportation Operations Insertion Algorithm

TPN Timed Petri Net

TR Total Rescheduling

TS Tabu Search

x_{iK} Completion time of last operation (K) of job i

x_{ikj} Completion time of operation k of job i on machine j

$$y_{ikprj} = \begin{cases} 1 \text{ if } o_{ik} \text{ follows } o_{pr} \text{ on machine } j \\ 0 \text{ otherwise,} \end{cases}$$

Copyright Notices

1) With kind permission from Springer Science + Business Media:

International Journal of Flexible Manufacturing System, Analysis of reactive deadlock-free scheduling in flexible job shops, Volume 19 (3), 264-285, Sherif Fahmy, Tarek ElMekkawy, and Subramaniam Balakrishnan, figures number: 1, 2, 3, 6, 7, 8, 9, and 10.

2) With kind permission from Taylor & Francis Group:

International Journal of Production Research, A generic deadlock-free reactive scheduling approach, *iFirst*, DOI: 10.1080/00207540802112652, Sherif Fahmy, Subramaniam Balakrishnan and Tarek ElMekkawy, figures number: 3, 4, 5, 6, 7, 8, and 9, and tables number 2 and 3.

3) With kind permission from Inderscience Publishers:

- International Journal of Operational Research, Mathematical formulations for scheduling in manufacturing cells with limited capacity buffers, *in press*, Sherif Fahmy, Tarek ElMekkawy, and Subramaniam Balakrishnan, figures number: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 and 11, and tables number 1 and 3.

- European Journal of Industrial Engineering, Deadlock-free scheduling of flexible job shops with limited capacity buffers, Volume 2 (3), 231-252, Sherif Fahmy, Tarek ElMekkawy, and Subramaniam Balakrishnan.

CHAPTER 1:

Introduction

1.1 Background

Functionality and capacity have been the two main parameters that defined the nature and requirements of a production system since the early times of the industrial revolution. In basic terms, functionality refers to the *variety* in product types that a single system can produce, while capacity refers to the quantities produced.

In the early twentieth century, the notions of mass production and Dedicated Manufacturing Systems (DMS) were introduced and established in North America, and the focus was primarily on the quantity produced. Due to the lack of competitiveness and the ever demanding war machine that prevailed during this era, DMSs had found their way to dominate the production globe. After the end of World War II, rising economical powers in Europe and Asia directed their attention towards industry, and hence more competitors were acquiring portions of the global market. In addition, the advancements in technology resulted in more consumer products being introduced into the market more frequently. This all resulted in a gradual shift in industrial focus from quantity to variety, and other types of more agile production systems being introduced.

With DMSs' dominant focus on quantity rather than variety, *job shop* systems have usually characterized the other extreme. A job shop system usually features a functional layout where machines that perform similar functions are grouped together in departments, and has usually been utilized to produce customized products. These systems are capable of producing *any* product whose manufacturing processes are within the capacity of the available machines. Parts flow through these systems according to their processing routes that define the sequence of operations required to complete the product, with no restriction on which machine (or department) can be visited next after completing an operation on another machine.

In a *general job shop* (will henceforth be just referred to as job shop), each product may have a different processing route (or direction of flow) through the system. A flow shop, on the other hand, is a special type of job shop that, like a DMS, has a uni-directional flow restriction. In other words, a part may enter a flow shop system at any machine, but has to follow the pre-defined direction of flow to complete all the required operations. Accordingly, flow shops offer a smoother and faster production pace, but without the full functionality provided by job shops.

By the mid 1960's, it was realized that approximately 60-80% of the discrete products market demanded mid-variety and mid-quantity products. To cope with this evident shift in market behavior, the notion of Group Technology (GT) emerged. The basic idea behind GT was to divide the massive capacity of a DMS between varieties of products. This demanded the modification of the general layout of the system from a production line, dedicated to produce only one product, to a group of production cells, and hence the

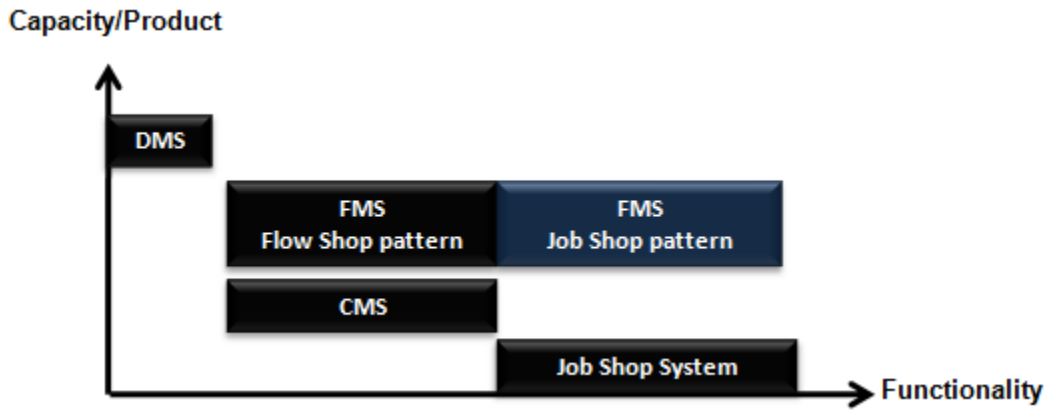
paradigm of Cellular Manufacturing Systems (CMS). To increase the functionality of the system, each of these cells was equipped with the group of machines (or processes) required to produce a *family* of products, rather than one product. Members of these families usually shared some common features related to the required manufacturing processes and the sizes of the products.

With the emergence of computers and the introduction of more advanced equipment in industry like Computer Numerically Controlled (CNC) machines, robot manipulators, and automated guided vehicles (AGV), these manufacturing cells gained their share of automation and were hence referred to as Flexible Manufacturing Cells (FMC). Consequently, the production paradigm featuring a group of FMCs supported by an inter-cellular material handling system was defined as a Flexible Manufacturing System (FMS). These systems have an inherent higher flexibility and functionality than their DMS counterparts and higher output capacity (quantities of products) than job shops as a result of applying GT and automation. These systems hence provide an acceptable balance between DMSs and job shops.

FMSs in industry usually feature a flow shop pattern. That is to say, members of a part family have a restricted uni-flow direction within a cell. With the high levels of automation that these cells feature nowadays, it has been argued that these systems include more functions than what is actually needed. An example would be the incorporation of a highly flexible robot to deliver parts to cater to unidirectional flow between machines rather than a simple conveyor that can accomplish the same task. However, the correct argument should be on how to utilize highly flexible equipment to

its functional extent to attain the sought variety, or better yet *flexibility*, of FMSs. The answer to this argument is the adoption of job shop flow patterns in FMSs. This will not only ensure the full functional utilization of the equipment, but also the flexibility of introducing any product to the system that does not necessarily require the same sequence of operations required by other members of a product family; hence higher functionality (Figure 1.1).

Figure 1.1: Capacity vs. functionality of the different production paradigms



1.1.1 Scheduling in Flexible Job Shops

Production scheduling is the problem of sequencing the parts (jobs) visiting the system to allocate processing times for their operations on the machines, such that some objective criterion is optimized. In the previous scheduling literature, the two most common objective criteria have been:

- Minimizing the total Makespan (MS): where the objective is to complete the processing of all the jobs in the system in the minimum possible time, and

-Minimizing the Mean Flow Time (MFT): where the objective is to minimize the average time spent by a job in the system.

The importance of these two criteria can be attributed to the positive impact they have on the system performance in general, and their secondary effects that simultaneously optimize other commonly used criteria. For example, minimizing the makespan subsequently yields higher machine utilization, whereas minimizing the mean flow time reduces the average work-in-process in the system and the average tardiness of jobs that have set due dates.

In a job shop each job can have a different processing route through the system and hence the scheduling problem becomes highly complex in systems that feature job shop flow patterns. Indeed, where the scheduling problem in flow shops is just that of finding the best sequence of jobs to visit the system, in job shops, the problem becomes that of finding the best sequence of jobs to visit *each machine* in the system. Furthermore, in more flexible job shops the problem is more complex since some (or all) jobs may have a number of *alternative* processing routes that can be alternatively followed to complete all the required processing operations.

1.1.2 Deadlocks in Automated Manufacturing Systems

Most automated manufacturing systems feature three inherent operational properties (conditions). These are:

- *Mutual exclusion*: jobs utilize system resources in an exclusive mode.

- *No pre-emption*: processing of a job on a machine cannot be pre-empted (interrupted) to process another job.

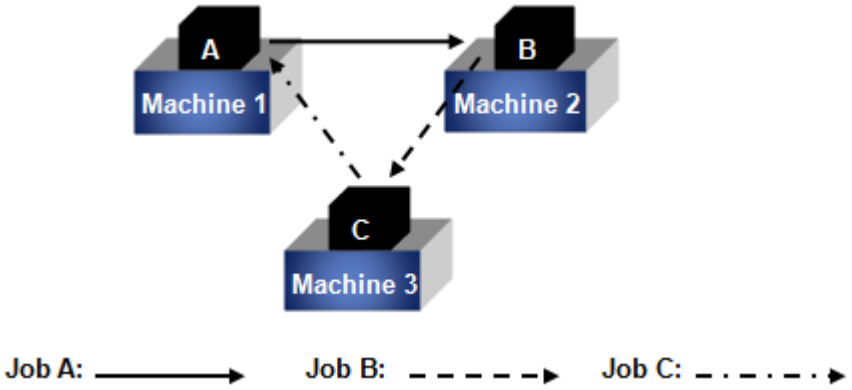
- *Hold-while-wait*: a job keeps holding (or blocking) a resource until the next resource in its processing route becomes available.

Because of these conditions and the inherent flow complexities in job shop systems, when they are automated they become highly prone to what is known as deadlock. A *deadlock* occurs in an automated manufacturing system when a set of jobs enter a *circular wait*. In this situation, each job in this set continues holding (blocking) a system resource indefinitely while waiting for another resource to become available, which is in turn held by another job in this same set (Figure 1.2). Eventually, the whole system or part of it becomes blocked, where no further processing could be accomplished, unless the circular wait is resolved. It should be noted that circular waits, as shown in Figure 1.2, cannot occur in flow shops because all the jobs share the same flow direction. Nevertheless, a deadlock can still occur in an automated flow shop if a material handler tries to deliver a job to an already occupied machine. These deadlocks, however, can be averted by simply ensuring that jobs get delivered to machines only when they become available.

In manually operated systems, circular waits can be easily resolved by human intervention by manually *swapping* the jobs between the machines or temporarily placing one of the jobs in a *buffer*. This is why assuming the existence of buffers with infinite capacities or simply ignoring the occurrence of deadlocks had been an inherent assumption when solving the classical job shop scheduling problem (to be discussed

further in Chapter 2). In automated (or flexible) job shops however, unless circular waits are prevented from occurring, or better yet the actual buffer capacity (if any) is taken into consideration when scheduling the jobs, deadlocks become inevitable.

Figure 1.2: Circular waits in job shops



1.1.3 Control in Discrete Event Systems

In discrete-product manufacturing systems, the transformation behavior from raw materials into finished products is almost dominated by discrete event activities. In other words, the system state changes at discrete points in time, only due to the occurrence of certain events. These can include the initiation of a processing operation, the completion of an operation, the delivery of a job to a machine, or the delivery of a job to a buffer. Accordingly, these systems are referred to as Discrete-Event Systems (DES). Automation of DESs essentially requires efficient controllability, which can be achieved by direct computer control of machine actuators at the local level and indirect computer control (or supervision) at the shop or *system level*. A *supervisor* can then be defined as a controller that uses available data via *feedback loops* to characterize the overall current behavior (or

state) of the system, and accordingly modify the lower level controllers via *control actions* to ultimately achieve the desired operational specifications in a deadlock-free manner.

Deadlock-free operation necessitates that the supervisor does not allow the occurrence of events that can directly or eventually drive the system into a deadlock state. This can be achieved either by:

- real time look-ahead to define the possible future states that can result from allowing an event to occur,
- pre-defining all possible deadlock states and preventing the occurrence of events that can drive the system into these states from other states, or
- only allowing the occurrence of the events that are known *beforehand* to drive the system to completion in a deadlock-free manner.

The first approach is not very efficient in real time control, especially when dealing with large systems where the number of future states that need evaluation is considerably large. Alternatively, although the second and third approaches can drive the system efficiently in real time, they are vulnerable to changes that frequently occur in dynamic production systems, which require frequent modifications in the control logic of the supervisor. Furthermore, while the first two approaches can be less restrictive from a control point of view, the third approach can be devised to not only drive the system safely (deadlock-free), but to also attain an optimized performance regarding the different production objectives.

1.1.4 Reactive Scheduling and Control

A production system, like any real world system, is subject to many uncertainties and disruptions. Frequent introduction of new production technologies, continuous changes in customer needs, large scale competition between producers, and the unforeseen internal disruptions that can affect the production process, all have led to a dynamic production environment. The occurrence of such disruptions can affect both the performance and *stability* of the production process. Hence, reacting to system disturbances is an essential part of any control system in an automated manufacturing environment. One that does not deal proactively with such disturbances can stall the whole system or render it in a chaotic state.

When a production schedule is initially set to allocate the processing times of jobs on the available machines, it consequently determines the utilization of other system resources, delivery dates of products, assignment of tools to machines, and many other production activities. Since these schedules are usually acquired in advance of the actual production to plan such decisions, upon implementation, they become subject to all these unforeseen randomly occurring disruptions. Hence, the scheduler part of the control system must be capable of modifying, or re-generating, the production schedule to account for such disruptions while preserving the efficiency and minimizing the amount of disturbance caused in the system.

When dealing with automated systems, such scheduling modifications have to be performed in real time. This consequently necessitates the availability of a robust supervisor that can reflect and implement such modifications also in real time. This

supervisor should hence be integrated with the reactive scheduler to ensure that the proactive reactions are taken while considering the overall performance and the amount of nervousness resulting in the system. Furthermore, it has to be ensured that the modified schedules and the re-developed supervisor can still drive the system safely without encountering deadlock states.

1.2 Research Motivation

The primary motivation for this research is to develop a control scheme that can be used to efficiently and safely drive FMSs that incorporate job shop flow patterns (will be referred to henceforth as flexible job shops). Although this type of system has a great potential to meet the demands of the current global market (Figure 1.1), the current industrial practice lacks a formal method by which these systems can be efficiently operated and controlled. As a result, most FMSs in the industrial practice use the flow shop perception and are still scheduled and operated using the simplest dispatching rules [1]. This in turn has led to the belief that these systems include more functions, and thus more capital, than what is actually needed, when the fact of the matter is that there functionality and flexibility are just underutilized. Developing a control scheme (or system) that can efficiently operate flexible job shops to their functional extent will not only provide more benefits to practitioners that have already adopted the FMS notion, but will further encourage more practitioners to exploit the benefits of implementing such systems.

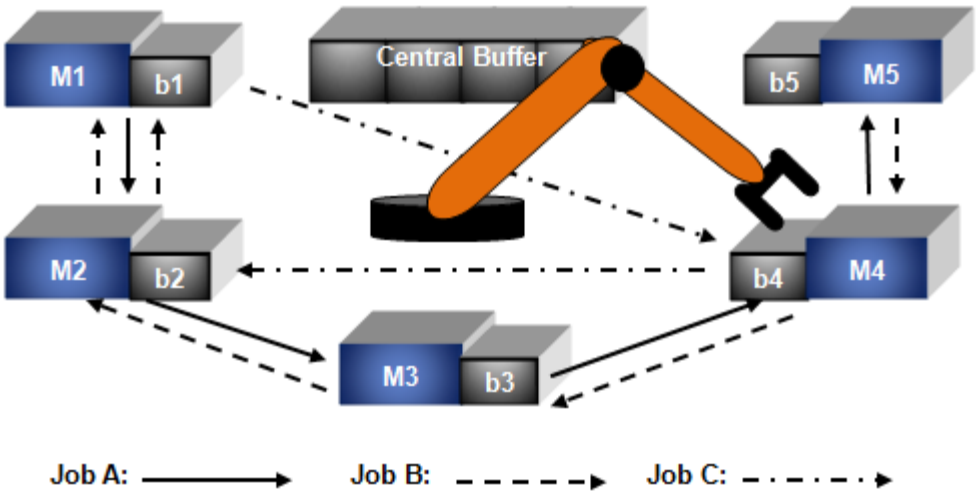
The secondary motivation for this research is to ascertain that not only flexible job shops can be efficiently operated, but are also robust enough to adapt to the ever changing nature of today's global market and any unplanned changes in the system. With the current unpredictable trends in the global product demand, it is becoming harder to forecast the best production levels that should be maintained for existing products. Moreover, the rapid pace in technological developments resulted in a high frequency of new consumer goods being introduced into the marketplace. The effects of these *external* variations are reflected on the shop floor in the form of new products being inserted suddenly in the production plan, due date revisions, lot size variations, and order cancellations. On the local level, the manufacturing system can also be a source of *internal* disruptions to the production process. Machine breakdowns and process time variations are common examples of such disruptions. Whether external or internal, when dealing with automated systems, the control system should be capable of incorporating and reacting to such disruptions in a deadlock-free manner, such that the overall performance of the system is minimally affected. Thus, if a robust controller for a flexible job shop is sought, it has to be designed accordingly.

1.3 Problem Statement

The type of systems considered in this research is automated flexible manufacturing systems (or cells) that feature a job shop flow pattern, or *flexible job shops*. An automated flexible cell usually comprises a number of CNC machines that are served by a dedicated material handler (like a robot manipulator). In addition, such cells usually include some

buffer capacity that can be used to temporarily store a job to preserve the continuity of flow or to resolve a deadlock. This buffer capacity can feature a central buffer that equally serves the whole cell or intermediate (input) buffers dedicated to the individual machines. A typical representation of such flexible cells is shown in Figure 1.3. The cell shown is comprised of five machines (M_1 through M_5) that are visited by three different jobs, A, B, and C. The figure shows the processing routes for the three jobs in a typical job shop environment. It also shows that the cell may have some buffer capacity in the form of intermediate buffers (b_1 to b_5) attached to the five machines, or a central buffer with an arbitrary capacity that serves the whole cell.

Figure 1.3: A typical flexible manufacturing cell



The following assumptions define the operational conditions of the considered system:

- Jobs use the machines and the transporter(s) in an exclusive mode.
- Job pre-emption is not allowed; i.e., the operation of a job on a machine cannot be interrupted to process another job.

- Each operation of a job has a fixed processing time on each machine that can process this operation.
- Some or all of the jobs may have routing flexibility; a number of alternative routes are available to choose from to complete the processing of a job.
- Set-up times of jobs on machines are independent of the sequence of visiting jobs and are included in the processing times.
- Transportation times between machines and other machines or the buffer are fixed.

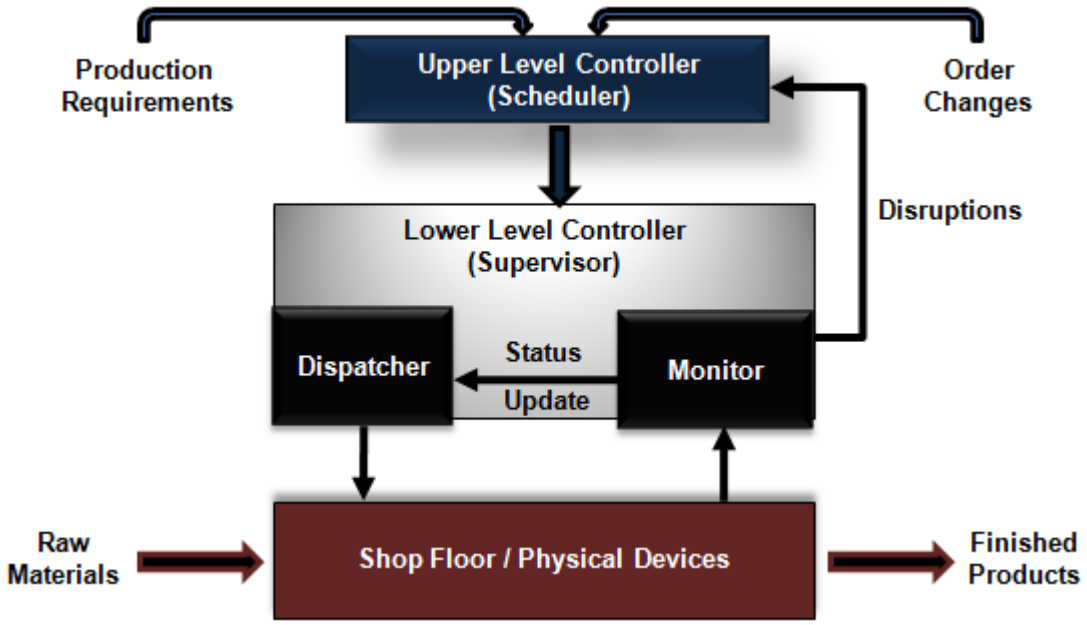
Because of the dynamic production environment in which these cells are intended to operate, they are usually prone to frequent disruptions. These can be external due to product mix changes, or internal, caused by breakdowns or variations in the manufacturing processes. Accordingly, the control system of such cells should be robust enough to absorb these disruptions and modify the control commands in order to maintain the stability and the optimized performance, while ensuring a deadlock-free operation of the cell.

The problem focused on in this research can thus be formally stated as follows: given an automated flexible manufacturing cell like the one illustrated in Figure 1.3, it is required to sequence and control the flow of the visiting jobs through the cell, such that the performance of the system is optimized and no deadlock situations are encountered. Furthermore, upon the occurrence of a disruption, it is required to maintain the performance and stability of the system, while preserving the deadlock-free operation, when reacting to this disruption.

1.4 Solution Approach

Ideally, the functions of a production control system can be classified into three distinct functional modules; a scheduler, a monitor, and a dispatcher. Accordingly, the current study proposes a hierarchical control system divided into an upper level scheduler, and a lower level supervisor that monitors and dispatches commands to the shop floor (Figure 1.4). The scheduler is responsible for determining a feasible allocation of the resources that optimizes some performance measure, based on the current production requirements and any unforeseen internal or external disruptions. The monitor collects and summarizes shop floor status information and feeds it back to the dispatcher and the scheduler. The dispatcher is then used to sequence and synchronize the physical activities in the system, based on the decisions of the scheduler and the feedback from the monitor.

Figure 1.4: Proposed hierarchical control system



Realizing the control system shown in Figure 1.4 can guarantee, not only the correct and safe operation of the controlled system, but also an optimized production performance as follows:

- The *scheduler* is the decision maker in the control system. According to the current product mix, it provides a *production schedule* that allocates processing slots for the jobs on the available machines while optimizing some production objective criterion. Taking into consideration the capacities of the available resources, this schedule will further ensure that the resulting job flow cannot cause any deadlock situations. Upon the occurrence of any internal or external disruption to the system, the scheduler will react to the disruption such that the updated schedule still retains the optimized performance with minimal variations from the original schedule.

- The *supervisor* is the command executer and observer of the system. In order to implement the original or the updated schedule on the shop floor, the assigned processing slots, and hence the underlying flow plan is *transformed* into a supervisory format that can interact with the shop floor devices. The supervisor will guarantee that the flow plan (behavior) determined by the scheduler is realized on the shop floor. It evolves in a discrete event manner, and is capable of receiving feedback signals and accordingly issuing action commands directly from/to the shop floor.

To attain and validate this hierarchical control design, the approach followed in this research can be detailed as follows:

1. Development of mathematical models for the deadlock-free scheduling problem of flexible job shops that can be solved to obtain optimal schedules

while considering a variety of system parameters. The models can then be utilized to schedule a new product mix for small systems, or in the design stages of medium ones.

2. Development of a heuristic, capable of solving the same scheduling problem for larger systems, which cannot be solved optimally using the mathematical formulations due to computational time limitations.
3. Development of a generic tool that can modify the production schedule in a deadlock-free manner to account for common internal or external disruptions in *real time*, while preserving the production performance and stability of the system.
4. Development of a formal method that can transform a production schedule into a discrete event supervisor in real time, which can realize the correct, optimized, and reactive behavior of the system determined by the scheduler.
5. Validation of the proposed hierarchical control approach by implementation in a real manufacturing setting.

1.5 Thesis Outline

This thesis is organized as follows. In Chapter 2, a comprehensive literature review on scheduling and control in manufacturing systems is provided, with special emphasis on deadlock-free scheduling and supervisory control in automated systems. The literature on reactive scheduling and control is also reviewed, and a number of controller implementation approaches are discussed. Chapter 3 provides novel Mixed Integer

Programming (MIP) mathematical models of the deadlock-free scheduling problem in flexible job shops with limited capacity buffers, and a novel operations insertion heuristic to solve the problem. Evaluations of both the approaches proposed are conducted via comparative analysis. In Chapter 4, the problem of deadlock-free reactive scheduling is addressed with emphasis on the product mix changes. A novel generic tool is proposed to revise the production schedule in the face of a number of internal and external system disruptions, while preserving the deadlock-free necessity in the revised schedules. A formal approach to realize a deadlock-free schedule through a discrete event controller (supervisor) is then proposed in Chapter 5. First, the deadlock-free schedule is transformed into a Marked Graph that captures all the precedence relations between the jobs on the different system resources. This graph is then augmented to include all the necessary components required to drive a system autonomously in a correct and optimized manner. Validation and evaluation of the hierarchical control approach via implementation in an experimental flexible manufacturing cell is presented in Chapter 6. Hardware and software components required to implement such a controller are also demonstrated. Finally, in Chapter 7, conclusions of this work and recommendations for future research directions are proposed.

CHAPTER 2:

Literature Review

2.1 Introduction

Efficient allocation of system resources and proper and correct realization on the shop floor can rather be expressed as scheduling and control, respectively. Scheduling and control of manufacturing systems have been widely researched and reported in the literature in the past decades. Although an ideal production control system should integrate the scheduling and the control tasks to ensure a correct and optimized behavior of the system, these two research areas have been treated separately in the past literature. This has lead to a gap between the contributions found in the scheduling literature and those pertaining to actual implementation on the shop floor [2], especially in automated manufacturing systems (AMSs). Furthermore, it was mentioned in [3] that despite the close relation between the control of AMSs and scheduling, there is a total lack of connection between the extensive literature reported on control of AMSs and those that deal with reactive scheduling. It was also mentioned that unless scheduling techniques consider the logic, or behavior of the system that should be realized by the controller, these schedules would certainly be unfeasible to implement. In fact, although much research effort has been devoted to production scheduling, most AMSs in the industrial environment are still scheduled using the simplest methods; namely dispatching rules [1].

Lately, to cope with the expanding trends in automation, deadlock-free scheduling and supervisory control approaches have, to some extent, dominated and replaced the corresponding traditional approaches. Although this has reduced the gap between the two hierarchical levels, the literature on deadlock-free scheduling still lacks a formal approach that can transform a schedule into an implementable supervisor, and the literature on supervisory control still considers the supervisor as the sole decision maker in the control system. A few attempts, however, have been made to integrate both levels, but these either lacked a global view of the system when performing the scheduling task [4], or realized a poorer performance via the supervisor when compared to schedules resulting from pure deadlock-free scheduling approaches [5].

The literature on scheduling and control in general is vast and too large to be covered in this survey. Accordingly, the focus here will be directed towards supervisory control and deadlock-free scheduling approaches, with occasional selections of some traditional scheduling approaches, mainly for the purposes of illustration and comparison. The literature on supervisory control can be classified into two main classes; approaches that utilized automata and those that utilized Petri nets (PNs). The objective in both cases was to acquire a supervisor capable of preventing the system from entering illegal (deadlock) states, while conforming to the technological and production constraints of the system. The traditional scheduling literature has utilized mathematical modeling, heuristics, insertion heuristics, priority rules, and meta-heuristics, amongst other approaches. Existence of infinite buffers has been assumed in all traditional approaches, and hence they cannot be adopted for AMSs, especially those that follow the job shop flow pattern

(Section 1.1.1). As a result, deadlock-free scheduling approaches have been receiving greater attention lately and are being pursued as a potential substitute.

The approaches proposed in the deadlock-free scheduling and the supervisory control literature are very closely related. In fact, these approaches have usually been interchanged in analysis. To account for system flexibility, specifically for system adaptability to unforeseen changes in the production state, the notions of rescheduling (or reactive scheduling) and reactive control have also been introduced to the literature.

In the sections to follow, the major supervisory control and deadlock analysis approaches will be reviewed first. This will be followed by a review of the deadlock-free scheduling approaches that utilized modeling tools like PNs and automata, which constitute the major portion of the deadlock-free scheduling literature. A review of some mathematical modeling and operations insertion approaches for the scheduling problem will then be presented. Literature on reactive scheduling and control will then be reviewed. Finally, a brief account on some controller implementation approaches will be given.

2.2 Supervisory Control of Automated Manufacturing Systems

The Supervisory Control (SC) problem for discrete event systems (DESs) can be formally defined as follows: given a plant (system) and the specifications of its desired behavior, the objective is to synthesize a controller that works in a closed-loop with the plant to ensure that it behaves legally according to the desired specifications [6]. The

word legally essentially implies the absence of deadlocks during the operation of the plant. For a production system, the specifications basically describe the modeled system with its resources and the processing requirements (or routes) of the parts (jobs) [7].

The Supervisory Control Theory (SCT) for DESs was proposed by Ramadge and Wonham (RW) [8, 9, 10] using automata and formal language models (Appendix B). In the SCT, the behavior of the DES is described by the sequences of events in the system. It is assumed that the system asynchronously generates the events. A supervisor is then defined as a controller that uses available data, or feedback, to characterize the behavior of the system, and disable undesired transitions in response [11]. These undesired transitions would lead the system to undesired states; mainly deadlock states. In general, a supervisor has three main tasks; i) monitoring the behavior of the system through feedback, ii) evaluating the current state and determining the appropriate control action, and iii) enforcing the specified control action [12].

2.2.1 Automata and Petri Nets

The development of a SC entails these three main steps [12]:

1. Modeling the DES (plant) and the specifications.
2. Synthesizing the supervisory controller with its control laws.
3. Implementing the supervisor and its control laws using a Programmable Logic Controller (PLC) or any computer based control system.

In the previous literature, several methods have been proposed for modeling and controlling DESs. These included RW's SCT, PNs, timed-transition models, real-time temporal logic, algebraic/language-based models and Moore automata. Although there exists no common agreement as to which one of these is the most effective modeling technique, the SCT and PNs have been the two most frequently used and commonly accepted methods by researchers for modeling and control of AMSs [13].

Some researchers agree that it is more advantageous to consider a SCT based approach over other approaches [14] because the resulting supervisors i) do not violate the specifications and are deadlock-free by construction, and ii) are optimal (maximally permissive) since all the legal events are allowed to occur in the system, unless they contradict with the defined specifications. On the other hand, other researchers find that, in automata models, the number of states increase exponentially with the system size and the graphical representation becomes unfeasible [15]. It was also stated in [11] that transforming system specifications into the formal languages of the SCT is not an easy task, and that analyzing such models for system performance is very difficult. In contrast, PNs were found to provide i) easily understandable graphical representations of the system and its specifications, ii) a well-formed mathematical basis, iii) a more compact description, and iv) analysis tools for the behavioral properties, performance evaluation and systematic construction of controllers (an introduction to PNs, their representational power, behavioral properties, and common structures is provided in Appendix A). However, it was stated in [16] that, unlike SCT-based models, optimal supervisors need not exist for all PN classes and that focus in PN models has usually been directed towards

proving the deadlock-freeness of the final model, with disregard to other important properties like reversibility.

There are some advantages and limitations associated with each of these two modeling techniques. However, PNs seem to provide a more convenient modeling tool for DES supervisors, and in the past decade PN approaches have, to some extent, dominated other modeling approaches. In the next sub-section, a review of the work that utilized and extended RW's SCT will be highlighted. This will be followed by a comprehensive review of the work that dealt with the supervisory control problem using PNs.

2.2.2 SCT Approaches

Centralized supervision implies that the whole system is supervised by a single controller that features the required closed loop behavior. Because of the aforementioned state explosion problem associated with automata models, the notion of modular supervision was introduced [10]. Modular (or de-centralized) supervision divides the system-wide supervisory task into two or more subtasks (or specifications). Each subtask is then solved using the SCT approach to obtain individual supervisors, which can be run concurrently to supervise the whole system [17]. Compared to the centralized supervisor, it is easier to obtain a modular supervisor because of the reduction in the number of associated states. Moreover, the modular supervisor is more readily modified, updated and maintained. The modular approach has been applied in many studies to control more complex AMSs than the ones controlled by the centralized approach. In [12], modular

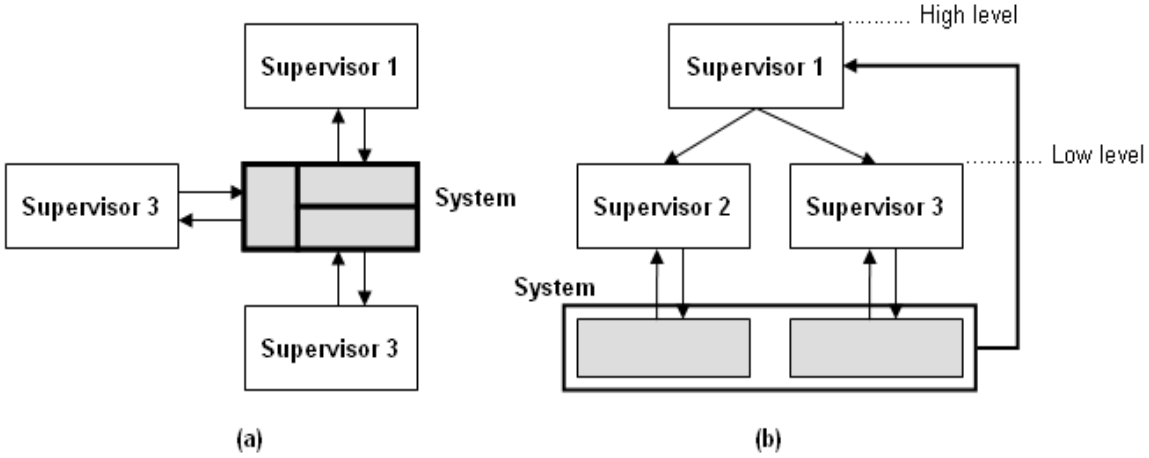
supervisors were employed to control a manufacturing cell consisting of a robot, a vision system, a conveyor belt, and two numerically controlled (NC) machines to produce two types of products. In [14], six assembly workstations working in series were supervised by three modular supervisors.

The reduction in state space achieved by modular supervisors comes at the expense of a considerably long computational time to ensure that the local supervisors are non-blocking (or non-conflicting). The idea of modularity implies that the control action of each of the modular supervisors is based on a partial view of the overall system state (Figure 2.1 (a)). Consequently, the control commands issued by these supervisors can sometimes come into conflict, especially when the behavior of some system resource is controlled by more than one supervisor [18]. Some studies proposed methodologies to introduce communication between the local supervisors [19], and to define conditional decisions that empower the local supervisors [20, 21] to enhance the performance. However, these methodologies in turn required substantial analysis and computational time.

While modular supervision implies a horizontal division of the control tasks, hierarchical supervision on the other hand implies a vertical division of these tasks [18]. In this setting, a high-level supervisor instructs and co-ordinates the tasks of lower-level supervisors. When more than one low-level supervisors are utilized, as is usually the case, it becomes a hierarchical-modular supervisor (Figure 2.1(b)). This is another form of de-centralization, which can simplify the analysis of complex systems using the SCT, while ensuring better coordination between local low-level supervisors. In [22], an

interface level was added in the hierarchy between the high and low levels, to obtain a hierarchical interface-based supervisory controller. This architecture enabled efficient analysis and verification of the global controllability and non-blocking properties of the overall supervisory system, just by verifying them at the local (lower) level. However, this approach still suffered from state space explosion when the synchronous product (Appendix B) of the high-level controller and all the interfaces is obtained. It was concluded that this problem may be overcome by considering more levels of hierarchy.

Figure 2.1: De-centralized supervision; a) Modular supervision, b) Hierarchical-modular supervision



The original SCT proposed by Ramadge and Wonham provided an abstract synchronization of the qualitative, or logical, behavior of the modeled systems, with no consideration of time. Temporal (timed) behavior was introduced in the SCT framework in [23] and [24] using timed transition models (TTM). This was achieved by introducing the activity transition function δ_{act} into the original generator automaton G, along with lower and upper time bounds within which events were permitted to occur. In addition, a

new event called *tick* was introduced into the model to represent the advancement in time, and the state space of the system had to be increased to correspond to the change in time. This approach has been applied within the frameworks of both centralized and modular supervisors. However, introducing timing into the model in the above manner, leads to a substantial increase in the number of states, and thus the complexity, of the system [18].

In [25], [26], and [27], the notion of time-augmented automata, or augmented finite automata (AFA), was proposed to incorporate the temporal behavior into the automata model. An integer global clock T, a variable C for each automaton that is reset to T every time an event changes the state of the system, and a *label* associated to each event to represent the earliest time this event may occur were introduced. The original generator G was then augmented with the finite set of labels Δ and a one-to-one function ω that maps each event to its associated label. This approach eliminated the increase in state space to represent the temporal behavior. It was also possible to model more complex manufacturing systems, namely job shops, and incorporate the temporal behavior of these systems into the model for cycle time optimization purposes. However, the results showed that obtaining the time-augmented models for relatively small job shops required much more computational time than other optimization approaches.

It can be concluded from the above discussions that the limitations of the SCT approaches can be attributed to the large state space required to represent even small systems, and the complexity of analysis of the formal languages. This lately has lead to the introduction of hybrid approaches that benefit from the advantages of both PNs and

SCT in controlling DESs [11, 16]. In these approaches, the modeling power of PNs was used to represent the uncontrolled behavior of the plant, and SCT was used to obtain the optimal (maximally permissive) supervisors for the plant. With the final closed-loop hybrid model described as a PN, PN analysis tools were then utilized to analyze the behavior of the modeled systems.

2.2.3 Deadlock Analysis and SC Approaches using PNs

Unlike automata models, PN models can readily and explicitly describe the behavior of the modeled system. Hence, the requirement of the supervisor to conform to the specifications of the system is inherently satisfied in the PN models. Accordingly, the SC problem in the PN formalism has usually been directed towards ensuring the safeness of the final model, or more precisely, guaranteeing the deadlock-freeness or liveness of the model.

In general, there are three strategies that address the deadlock problem; deadlock *prevention* methods, deadlock *avoidance* methods, and deadlock *detection and recovery* methods. Prevention methods guarantee that deadlock conditions cannot be simultaneously satisfied at any point during the operation of the system [28]. They are usually applied offline during the design stage, and hence do not require the knowledge of the system state to realize the control action. Avoidance methods, on the other hand, are online methods that require the knowledge of the current state of the system in real time. Based on the system state, the controller can then inhibit or enable events using

look-ahead procedures that determine future deadlock states. These methods also have to detect unsafe states that may lead the system inevitably to deadlocks. Finally, detection/recovery methods employ a monitoring mechanism to detect deadlocks and a resolution procedure that can terminate deadlocked operations by releasing the associated system resources. However, this strategy is difficult to apply in fully automated systems, and can result in considerable system downtime if deadlocks are frequent.

In the PN literature, deadlock avoidance and prevention policies have usually been the ones adopted. Some approaches proposed pure avoidance or prevention policies, while others often proposed hybrid strategies that define some prevention control strategy offline and apply this strategy online to avoid deadlocks. Furthermore, the PN literature on deadlock avoidance and prevention can be fairly classified into generic approaches that considered the liveness of the whole net, and approaches that characterized the deadlock states using siphons analysis.

2.2.3.1 Generic PN deadlock analysis approaches

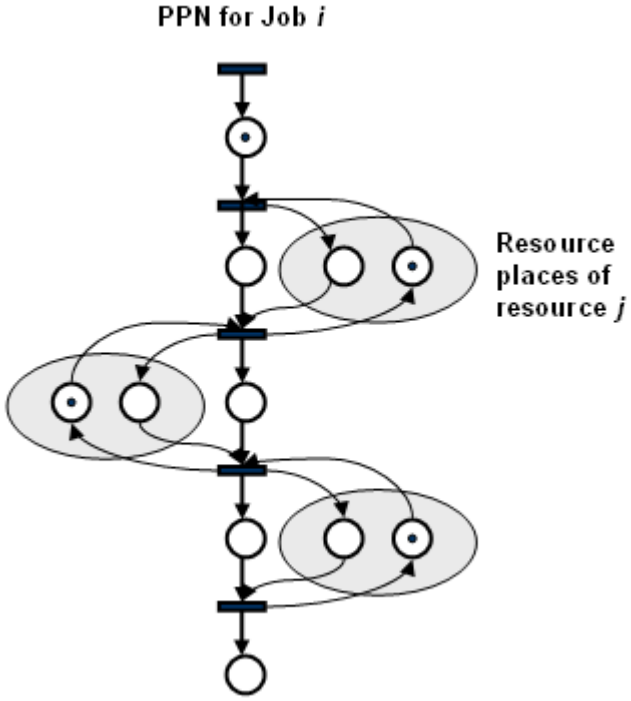
In [29], deadlock prevention and deadlock avoidance techniques for PNs were proposed. The deadlock prevention policy was based on the *exhaustive* search of the reachability graph (RG) of the PN model to identify deadlock markings, and the determination of the paths in the RG that do not lead to those markings. On the other hand, the avoidance policy constructed the RG only for a finite number of look-ahead steps from the current state, to check for potential deadlock markings. The policy then

avoided the undesirable paths by restricting the firing of the transitions to safe paths only. However, approaches based on the reachability analysis suffer from state space explosion when the system size, and hence the number of states, increases.

In [30], a deadlock avoidance algorithm (DAA) was proposed for AMSs. In addition, the PN structure, *Production Petri nets* (PPNs), was defined to model the manufacturing systems. A PPN consists of a set of shared resource places, where each resource is represented by two places; tokens in the first place represent the jobs holding the resource and those in the second represent the available number of units of this resource (Figure 2.2). The production sequence of each job (part type) is then represented by a sequence of places and transitions, where places represent the required steps to produce the job and the transitions model the release or acquisition of a resource to complete the associated step (or operation).

The proposed DAA was defined as a resource allocation restriction policy that selects specific enabled transitions for firing. This policy was based on two rules: i) a token is allowed to enter a new zone in the production sequence only when the capacity (number of available resource units) in the shared sub-zone exceeds the number of tokens (jobs), and ii) if a shared resource is requested by a job, all the other resources in the associated zone must be available. This policy was relatively simple because it was based only on the analysis of resource requirements in the current production sequence zone for each job. However, it was too conservative because it allowed a limited number of jobs in each zone, and hence resulted in a poor overall system performance.

Figure 2.2: A production Petri net (PPN)



In [31], another deadlock avoidance policy was proposed for an extension of PPNs that allows an operation on a job to acquire more than one resource simultaneously. The policy was achieved by augmenting the PPN with control places, defining a controlled production Petri net (CPPN). The function of the control places in the CPPN is to incorporate exogenous conditions for disabling transitions, whose firing may lead to a deadlock. A validity test was also proposed to ensure the liveness of the CPPN for a given initial marking, by determining the sequence of markings that lead to the completion of all the required operations. Like the policy proposed in [30], the avoidance policy proposed in [31] was too restrictive [28]. In [32] and [33], deadlock prevention techniques were proposed for PN models of AMSs with shared resources. The PN models were synthesized and analyzed to ensure structural liveness and reversibility. This was

achieved by adding buffer modules or limiting the number of parts in the process, which increased the conservativeness of the model.

In [15], the notion of Automation Petri nets (APNs) and the inhibitor arc method for the control of DESs were proposed. In APNs, the firing of each transition is associated with the occurrence of an external event. In addition, firing a transition initiates an event in the system, like the start of a machining process. The inhibitor arc control method involved first representing the uncontrolled behavior of the system with an APN. The RG of the APN was then generated, and the bad markings (deadlock states) were identified and removed. The reduced RG was then transformed into another APN, which was connected to the original APN with inhibitor arcs to prevent the occurrence of events (firing of transitions) that can lead to the identified deadlock states. In [16], instead of generating the RG of the uncontrolled APN, the SCT was applied to obtain the maximally permissive RW supervisor of the system. The obtained RW supervisor was then transformed into an auto-net, which is a PN-like representation, which was then connected to the APN model of the plant with inhibitor arcs.

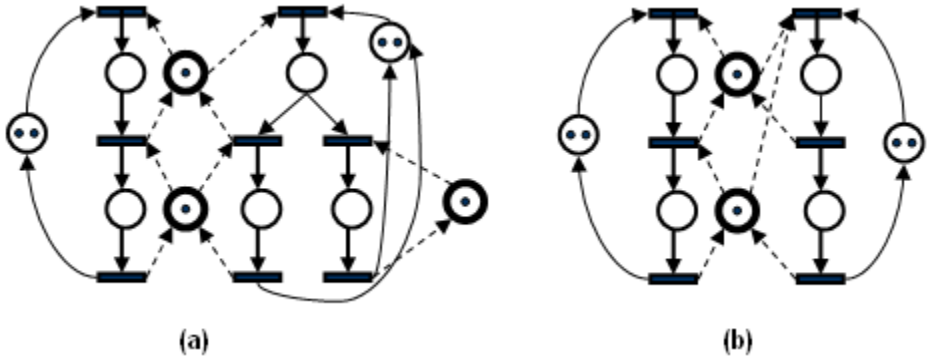
2.2.3.2 Siphons and deadlock analysis

The above deadlock avoidance and prevention approaches suffered either from the state explosion problem or the restrictiveness of the PN model. This was due to the absence of a characterization of deadlock states [28]. Deadlock states in PNs can be directly associated with the presence of empty siphons in the net (Appendix A). In [34],

PPNs were extended to model flexible routing in the system, and a new PN structure referred to as Systems of Simple Sequential Processes with Resources (S^3PR) was defined. An S^3PR is a set of cyclic state machines (SMs), each with an idle place having a number of tokens equal to the number of parts to be processed. These SMs are connected via a number of places with tokens representing the number of available units of each resource. The liveness of these nets was analyzed and it was shown that a S^3PR net is live iff for each reachable marking each minimal siphon had at least one token. A control policy that determines the set of minimal siphons that can be emptied was established, and additional places were introduced to the net to prevent these siphons from becoming empty.

In [35], the liveness of another class of PNs was proved using minimal siphons. The class of augmented marked graphs (AMGs) was introduced, and it was proven that an AMG is live and reversible if no siphon is a potential deadlock (can become empty). AMGs are cyclic marked graphs (MGs) augmented with places that represent the available resources in the system. The main differences between AMGs and S^3PR nets is that in the former, a processing stage can utilize more than one resource, and in the latter, routing flexibility is allowed (Figure 2.3). A sufficient condition for liveness based on the state equation of the net was also proposed, and it had to be verified for each minimal siphon that does not contain a marked trap. A Mixed Integer Programming (MIP) model was further formulated to check for potential deadlocks.

Figure 2.3: Special PN classes: a) S^3PR nets; b) Augmented marked graphs (AMGs)



Because the number of minimal siphons is exponential with the number of places in the net, the control policy proposed in [34] had an exponential complexity. This problem was addressed in [36], where a logic programming algorithm was proposed to find the set of minimal siphons in S^3PR nets. This algorithm had a fast convergence rate, but still suffered from an exponential worst-case complexity. In [37], S^3PR nets were extended to model jobs that can utilize more than one resource simultaneously by defining a PN structure called Systems of Sequential Systems with Shared Resources (S^4R nets). A deadlock prevention policy that augmented the net with *local* additional places for each minimal siphon was proposed to ensure that the net will remain marked for all reachable markings. A look-ahead avoidance policy that did not guarantee the deadlock-freeness in every case was also proposed. Alternatively, if a deadlock state was reached, a recovery procedure was initiated to resolve the deadlock.

In the subsequent years, some approaches have been proposed to extend the types of systems that can be modeled using similar PN structures, where deadlocks can be characterized using minimal siphons. For example, in [38], the analysis of S^3PGR^2 nets

was proposed. These nets can model systems where every processing stage poses a finite number of alternative requests, such that each requires an arbitrary number of units of each resource from a set of resources. Other approaches proposed more efficient algorithms to compute the minimal siphons for special classes of PNs [39]. However, the computation of minimal siphons remained with an exponential complexity.

Another problem with the minimal siphons approach for deadlock prevention is that the control policy adds an additional place to each minimal siphon in the net to ensure its liveness. This results in a much more complex PN control model than the original PN model of the system [40]. To overcome this problem, the notion of *elementary* siphons was introduced in [41]. In this study, it was shown that not all the minimal siphons in the net need to be controlled (associated with additional control places), but only a limited set of elementary siphons. It was also shown that the number of these elementary siphons is only bounded by the number of places or transitions in the net (whichever is smaller). In [42], an algorithm that can compute the optimal set of elementary siphons in polynomial time was proposed. However, it was stated that this optimal set cannot be computed for all PN structures. In [41], the algorithm was applied to S^3PR nets to identify this optimal set of siphons in polynomial time. However, a shortcoming of the elementary siphons approach is that it required the prior knowledge of all the siphons in the net in order to identify the set of elementary ones. Hence, it also suffered from the computational complexity associated with computing the siphons of the net.

2.3 Deadlock-free Scheduling

In the context discussed in the previous sections, the task of a controller (supervisor) in an AMS was to realize the correct behavior that grasps the requirements of the system. In another context, the role of the supervisor is just to coordinate and synchronize the activities within the AMS [43]. In the latter context, a higher-level scheduler determines which job the AMS accepts next, which machine a job visits next, and which job a machine processes next. Indeed, it was also mentioned in [2] that the scheduler should be the only decision maker in the overall control system. Employing such a hierarchy will not only guarantee the correct behavior of the system, but also generate the one that is optimal. Although much research has been devoted to production scheduling in the previous decades, most production systems in industry still depend on very simple dispatching policies for scheduling [1]. This, especially in AMSs, can be mainly attributed to the lack of deadlock consideration in the scheduling level.

The research on production scheduling can be traced back to the beginning of the previous century. This has included mathematical modeling, dispatching-rule-based-heuristics, simulation, fuzzy logic, neural networks, local search algorithms, and met-heuristics [26]. However, the notion of deadlock-free scheduling has begun to attract attention only in the mid nineties. This was due to the increasing interest in AMSs, the wide-spread of new manufacturing paradigms like FMSs, and most importantly the introduction of the SCT and the application of automata and PNs in characterizing deadlocks. Because automata and PNs have been the most widely employed modeling

formalisms to detect and/or characterize deadlocks, they have usually been combined with conventional scheduling approaches to obtain deadlock-free schedules.

2.3.1 PN and Automata Scheduling Approaches

The work proposed in [44] was perhaps the first to introduce the technique of searching the RG of a Timed PN (TPN) to find the best deadlock-free schedule. In the TPN, operation times were associated with places. The method first formulates the scheduling problem using a PN model that captures the properties of the system to be scheduled. A *partial* RG is then generated, based on a heuristic function, to define the best schedule in terms of a sequence of transition firings. The generated firing sequence drives the net from its initial state M_o to a final state, where all the jobs have completed their processing in the system with the best makespan (MS). Potential deadlock states are avoided in the path generation process since they are depicted by transitions that cannot be fired. To avoid searching the whole RG, an A^* search algorithm was employed. The A^* search algorithm is a best first (BF) strategy, where each node in the search tree is assigned an optimistic estimate of the 'cost-to-go'; that is, the cost (time) of reaching a node representing a final state. The nodes with the best cost estimate were then selected for further exploration. The presence of intermediate buffers between machines that can resolve deadlocks was also considered. It was concluded that formulating the scheduling problem using PNs can easily handle essential properties of FMSs, like alternative routing, shared resources, and different lot sizes.

In [45], a hybrid search algorithm was combined with TPNs to find the best schedules, with no consideration for deadlocks. Infinite buffers were assumed to exist in the system. In this approach, a combination of a BF strategy and a Backtracking (BT) strategy was utilized. The role of BT was to improve the solution found by the BF search. When a final marking was reached, the BT was initiated up to a stage where alternative enabled transitions existed. It then explored a new (alternative) branch by firing one of the alternative transitions and repeating the BF strategy to obtain a new solution. In [46], this work was extended by considering limited intermediate buffers and material handling operations. Places representing buffer capacities were introduced into the TPN such that, a job was only allowed to move to the next machine in its processing route if this machine was available or the intermediate buffer preceding it had some available capacity. In [47], the hybrid BF-BT search procedure was utilized to schedule systems with alternative routes for jobs. A new heuristic function to provide better cost estimates in the search process was also proposed.

In [48], a hybrid BF-BT search procedure was again employed, but with embedded priority rules to select from possible alternative enabled transitions. The least work remaining (LWKR) rule was used, and in case of transitions having the same LWKR, the shortest processing time (SPT) rule was utilized to break the ties. Timed S^4R nets were used to model the system and the approach developed in [36] was utilized to compute the minimal siphons of the net. The liveness condition of S^4R nets, which is related to minimal siphons, was then used as a truncation technique to reduce the search space, and consequently the computational time of the algorithm. In [49], this work was extended by considering the mean flow time (MFT) as the scheduling objective criterion rather than

the MS, which was considered in all previous studies. A user control factor was also proposed to compromise between the quality of the obtained solution, and the computation time needed to acquire the solution.

In [50], TPNs were again employed to model the manufacturing system. While a cost function was calculated to select the transition to be fired at each state, the algorithm was set to look ahead several transitions to ensure that deadlock states are not reached. Deadlock states were defined as states where the number of jobs in the whole system is greater than or equal to the limited capacity of a central storage device (buffer) that serves the whole system. The cost function was evaluated using the SPT or the LWKR priority rules.

In [51], the transition sequence of the TPN model of the system was coded with natural numbers as chromosomes in a genetic algorithm (GA). Siphons analysis was employed to detect transitions leading to deadlocks, and was incorporated as a deadlock-free constraint in the GA. Upon the detection of a bad transition, a decoding scheme was used to reschedule the transition sequence to avoid the deadlocks. Finally, a penalty item was added to the fitness function of the GA to avoid converging to unfeasible solutions. This work was extended in [52] by considering multiple resource allocation systems, where a part can use several units of several types of resources at a single processing step.

In [53], a TPN model was used to represent the system, and a beam search (BS) strategy was associated with the execution of the net. In a BS strategy, at each search step, a number of nodes equal to the set *beam width* are evaluated. In order to avoid

deadlocks, a siphon detection algorithm was employed to check whether the addition of a node to the beam would cause a deadlock. The PN generation and siphons detection algorithm were employed off-line to determine the deadlock states.

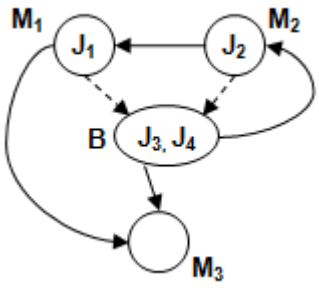
In [54], a feasibility condition for deadlock-free schedules was proposed. Schedules were represented by MGs, and it was concluded that a schedule was feasible if it was *free from elementary circuits*. The notion of simple systems was also introduced, where resources are released as soon as the processing of a job is completed; in other words, the hold while wait condition was not considered in simple systems. A dynamic programming algorithm was also proposed to solve the scheduling problem of simple systems to optimize the MS. In this algorithm, the scheduling problem was solved recursively as a one-job problem, until all the jobs are included in the schedule. In [55], the hold while wait condition was incorporated in the dynamic programming algorithm. The schedule was represented using a place timed MG and a condition was added in the dynamic program to check if performing an operation at a certain time period would lead to a deadlock.

As for automata approaches, in [56], timed automata were employed to represent manufacturing systems with limited buffers and an A* tree search algorithm was used to find the best schedule. The search for deadlock states was done off-line, before the application of the A^* algorithm and these states were removed from the search tree to reduce the search space. In [25] and [27], the RW supervisor was obtained for the modeled system using automata formalism. In order to search for the best schedule, the RW supervisor was augmented with time to obtain the AFA (Section 2.2.2), and a BF

strategy was applied to determine the deadlock-free schedule with the best MS. In [26], a beam search (BS) strategy was employed instead of the BF strategy to find the best MS. A *k*-step-look-ahead evaluation was also employed, such that *k* determines the depth (or number of look-ahead steps) of each iteration. It should be noted that, before using both search techniques (BF & BS), the AFA and the RW supervisor were obtained off-line to reduce the computation time.

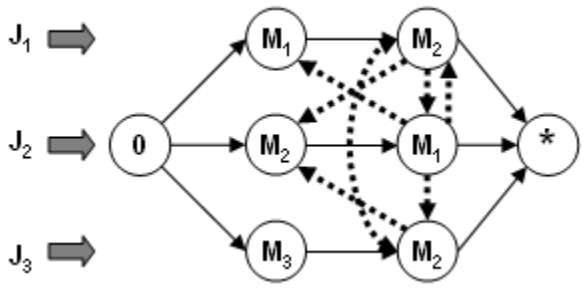
Although most of the studies concerned with deadlock-free scheduling relied on the deadlock characterizations provided by PN and automata structures and analysis methods, a few approaches considered other formalisms to define deadlock states. In [57], the presence of central buffers was considered, and a GA was embedded with the transition sequences of a graph theoretic approach employing transition digraphs (TD) to optimize the deadlock-free scheduling problem. The TD represented the information regarding job-resource interactions at each state of the system. The nodes of the TD represented the resources (machines and central buffer), and the directed arcs represented job transitions from one resource to another (Figure 2.4). A deadlock state was then defined as a state where every reachable node is capacitated (occupied resource).

Figure 2.4: A transition diagraph (TD)



In [58], Tabu search (TS) was combined with an extension of the classical geometric approach, which was previously used to solve the classical two-job problems, to obtain deadlock-free schedules. In order to detect deadlocks, a modification of the disjunctive graph (DG) representation was used to model the schedule (DG with blocking), in which the hold while wait condition was respected. In the proposed DG, each processing operation was represented by a node and two types of arcs were defined to connect these nodes; conjunctive arcs and disjunctive arcs. Conjunctive arcs connected the consecutive operations in a job's processing route to ensure the precedence constraints, while couples of disjunctive arcs connected the operations sharing the same resources, such that the hold-while-wait condition was respected (Figure 2.5). The TS algorithm was used to select one of each of the disjunctive couples to obtain a schedule. A deadlock was then defined as a circuit of disjunctive arcs in the DG. Upon the detection of a deadlock, a job was rescheduled using the geometric approach to resolve it.

Figure 2.5: The disjunctive graph (DG) with blocking



Finally in [59], the modified geometric approach was used again to obtain deadlock-free schedules. However, in this second study, a deadlock detection algorithm incorporated in the geometric approach was used instead of the disjunctive graphs method. The geometric approach was further modified to schedule the jobs in the system one after the other, such that the sequences of already scheduled jobs are not altered when a new job is added to the schedule. The geometric approach considered all the previously scheduled jobs as one job, and hence solved a two-job problem every time a new job was added to the schedule. The deadlock detection algorithm was imbedded in the geometric approach, and had a polynomial computational complexity in the number of jobs and number of operations on machines. Deadlocks were defined as obstacles in the modified geometric approach that must be avoided by the feasible path representing the schedule. These obstacles represented cases of conflicts between resources operating on the two jobs and cases of deadlocks.

2.3.2 Mathematical Modeling in Scheduling Job Shops

Mathematical modeling (programming) is a typical traditional approach for modeling production scheduling problems [60]. In the past literature, the traditional job shop scheduling problem has often been solved while assuming the existence of buffers with infinite capacities. The scheduling problem in general is classified as NP-hard from the computational complexity point of view [61]. As a result, mathematical models have been hardly proposed to solve the *deadlock-free* scheduling problem, since it is even harder than the traditional one. However, due to the everyday advancements in computer capabilities and the existence of highly efficient commercial software for modeling and solving linear and non-linear programming problems, solving complex mathematical models has become possible [61, 62, 63].

In general, solving a mathematical model of a problem using an exact method provides the optimal solution. In addition, mathematical models provide general framework for the considered problems that can usually be adapted to many instances by adding or deleting some constraints [63]. In [64], an MIP formulation was proposed to solve the traditional job shop scheduling problem. The models proposed in [61], [65], [66], [67], [68], and [69], did not consider the hold while wait condition, but considered some flexibility related issues like alternative routes, parallel resources of the same type, and re-entrant jobs (or operations). In these models, routing flexibility has been represented using two main approaches. In the first, a job was assumed to have more than one alternative process plan, where each was assigned an integer variable with the role of indicating whether that process plan was selected or not [65,66]. In the second, an

operation of a job was allowed to select its processing machine from a pre-defined set of machines. This selection was again modeled by binary decision variables [68].

Solving the deadlock-free scheduling problem of job shops using mathematical models was only considered in [70]. In that study, the deadlock-free scheduling problem was modeled by two mathematical formulations. In the first model, it was assumed that the modeled system had no buffer space, while in the second, intermediate buffers were assumed to exist between the machines. However, the second model did not lead to optimal schedules since buffer capacities were not represented in the model. In fact, it was mentioned in [46], [54], [58], and [70], that representing capacities of buffers of a job shop system in mathematical models is a very difficult task.

2.3.3 Insertion Heuristics

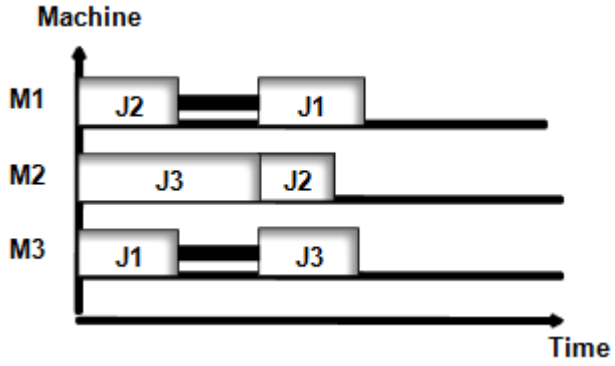
Operations and job insertion scheduling approaches have been applied in previous literature with two different objectives; to insert newly arriving jobs, or to build schedules of all the available jobs. In [71], a polynomial insertion algorithm, based on disjunctive graphs, was proposed to solve the job shop scheduling problem with multi-resource requirements and resource flexibility, and the on-line insertion problem of new jobs. This work was extended in [72] by considering jobs with sequence-dependant set-up times. In [73], a real time insertion algorithm based on dispatching rules was proposed to schedule continuously arriving new jobs on a single machine. In [74], MIP models were proposed

to solve the job insertion problem, and a relaxation method to compute a strong lower bound for the problem was proposed.

In [75], a BS job insertion algorithm was proposed to solve the traditional scheduling problem of jobs with sequence-independent setup times. Schedules were generated and checked for feasibility by utilizing the *rank matrix* representation of the schedule. This rank matrix is equivalent to the special Latin rectangle (*LR*) that was proposed earlier in [76]. The idea behind a rank matrix was that it provided all the routing and sequencing information of a schedule in one compact form. The rows of the matrix provided the processing routes of the jobs, and its columns provided the sequences of jobs visiting each machine. Like the above approaches, the utilization of rank matrices in [75] and [76] was to solve the traditional scheduling problem with no consideration for deadlocks.

Although insertion algorithms usually outperform algorithms based on priority rules [75] and those based on enumeration algorithms [72], it was noticed that the deadlock-free scheduling problem was hardly approached using insertion algorithms. In [77], however, an insertion algorithm based on disjunctive graphs was proposed to solve some extensions of the traditional job shop problem, including the blocking job shop problem. In this problem, no buffer space was available, and the time within which a job could hold (or block) a machine until the next machine in the processing route becomes available was considered. However, these considerations do not prevent the occurrence of deadlocks, since circular waits can still be formed as shown in Figure 2.6.

Figure 2.6: Blocking job shop schedule



In Figure 2.6, job J_1 is not allowed to start processing on machine M_1 until J_2 releases M_1 and starts processing on M_2. Hence, the time during which a job blocks a machine is considered. However, unless there is some buffer space where one of the three jobs can reside, the shown circular wait between J_1, J_2, and J_3 will practically prevent any of these jobs from moving from one machine to another, and force the system into a deadlock.

2.4 Reactive Scheduling and Control

Although in [78] seventeen types of production system disruptions were mentioned, most of the reactive scheduling literature has focused on two main types of disruptions; arrival of new jobs [79, 80, 81, 82] and machine breakdowns [83, 84, 85, 86, 87, 88]. However, a few studies have considered other types of disruptions, often along with the previous two, like process times variations [89, 90], order cancellations [91, 92, 93], and due date changes (or urgent jobs) [90, 91].

Revising a production schedule upon the occurrence of a disruption affects both the efficiency and the stability of the production process. The effect on the efficiency is usually measured in terms of the percentage change in the scheduling objective criterion between the original and the revised schedules. Stability has often been measured in terms of deviations in the starting times of operations between the two schedules [78, 81, 83, 90]. Unfortunately, pursuing the optimum efficiency of the revised schedule usually results in major deviations from the original one, and vice versa. For instance, a total rescheduling (TR) approach solves the problem from scratch for the remaining operations in the schedule, with the objective of optimizing the original scheduling objective criterion, with no regards to the resulting disturbance [94]. On the other hand, the Right Shift Rescheduling (RSR) approach aims at minimizing the amount of deviation caused in the schedule by simply pushing all the remaining operations ahead in time to compensate for the disruption [85]. This is done without changing the defined sequences of jobs on the machines. In between these two extremes, many schedule repair approaches have been proposed in the past literature. These included heuristics (78, 79, 83, 90, 95], dispatching rules-based heuristics [80, 96], filtered BS [93], GAs [81, 91, 92], Tabu search [97], and artificial intelligence [98].

In [83], the Affected Operations Rescheduling algorithm (AOR) was proposed to overcome the deficiency of the RSR approach when encountering a machine breakdown in the system. This algorithm produced more efficient and stable reactive schedules when compared to RSR. AOR was later modified in [86] to account for batch processing changes and in [82] to account for new urgent orders. In [78] and [90], AOR was used as a core for a modified AOR (mAOR) heuristic that defined generic schedule repair actions

for a wide variety of disruptions. In the first study, mAOR was compared to RSR regarding the performance for four types of disruptions; machine breakdowns, arrival of new orders, process time variations, and urgency of existing jobs. The results showed that mAOR outperformed RSR in terms of both efficiency and stability.

Reactive schedules that do not recognize situations like blocking of machines and deadlocks, upon implementation through the controller will eventually drive the system into these illegal states. However, only a few reactive scheduling studies considered the deadlock-free operation of the system. In [89] and [99], hierarchical dynamic rescheduling systems were used. In these systems, when deadlocks were detected, a controller (or process runner) module was used to avoid it. In [92], adaptive GAs were used for dynamic *total rescheduling*, and when the GA solution yielded sequences that cause deadlock situations, simple dispatching rules were used to modify them to ensure the feasibility of the schedules. In [84], deadlock situations were prevented while considering finite capacity buffers. However, only machine breakdowns were considered in that study.

Finally, if a deadlock-free disrupted schedule is input to RSR, AOR, or mAOR, the resulting revised schedule would be also deadlock-free, since pushing operations ahead in time does not create new circular waits between existing jobs. However, although the AOR could be efficiently applied to react to machine breakdown disruptions while keeping the deadlock-free status of the schedule, its application through mAOR to react to other disruptions, like arrival of new jobs, may lead to considerable deterioration in both the efficiency and stability of the resulting schedule.

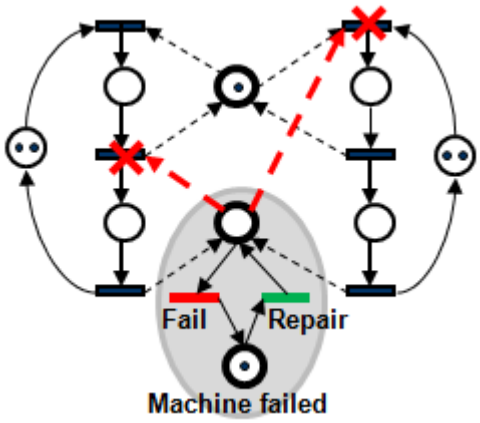
2.4.1 Reactive Supervisory Control

As mentioned earlier, the approaches for designing a supervisory controller for a production system, using automata or PNs, strive to ensure that the controller is maximally permissive. This means that the controller would allow any event to take place in the system as long as it does not violate the production requirements or lead to a deadlock situation. Accordingly, the occurrence of an internal disturbance in the system, like a machine breakdown, would simply require that some events should be disabled (failed machine should not be loaded). The system can then proceed by executing any of the allowed events (if any), until the source of disturbance is alleviated (machine is repaired).

To react to machine failures in the automata formalism, instead of just representing the states of a machine as idle or busy, other states are added to the model to indicate that the machine is in a failed state. Upon repair, its state is changed back to idle, allowing its associated events to be executed. This approach however, increases the complexity of the model and the computational time required to obtain the supervisor model dramatically. In [7], an approach employing two algorithms was proposed to react to machine failures. The first algorithm disabled the events requiring the utilization of the failed machine. The second algorithm determined the allowable events after the repair, in accordance with the status of the part that was being processed on the machine when it failed (if any). It was shown that the computation time required to execute both algorithms was not significant because they did not add any new states to original model. In the PN formalism, indicating that a resource has failed simply requires removing the token in the place

representing the availability of the resource. In this case, all the output transitions of this place will be disabled, and tasks requiring the failed resource will not be executed (Figure 2.7).

Figure 2.7: Resource failure in a PN



In the case of external disturbances, like the arrival of a new job (order) to the system, the problem becomes much more complex. The automata or PN model of a production system, especially job shops, depends to a great extent on the processing requirements of the available jobs. Upon the arrival of a new job that has new requirements, the whole model must be reconfigured in order to capture these requirements. This in turn requires calculating the supervisor model, whether using the SCT in case of automata or siphons analysis in case of PNs, to obtain the new correct behavior of the updated system. In both cases, as mentioned earlier in this chapter, this requires a substantial amount of calculation time that does not concur with the requirements of real time reactive control.

In [13], this problem was addressed using the SCT and automata formalism. The idea of a supervisor pair was proposed, where a nominal supervisor is ordinarily developed for

the original parts in the system, and a complementary supervisor is added whenever a new job is added to the system. The function of the complementary supervisor(s) was to control the requirements of the new job(s), while preserving the structure of the nominal supervisor and avoiding computing a new controller for the whole system. Although this approach requires less time than updating the original supervisor, it was mentioned that it still required a substantial amount of time to calculate, especially when the new parts interact with many machines in the system. This approach also suffers from the requirement of installing new hardware and wiring for the complementary supervisors, every time a new order arrives.

The reaction to other system disturbances has not been considered in the SC context. The cancellation of an order, like machine breakdowns, can simply be treated by disabling the events associated with the cancelled job. As for performance related disturbances, like process time variation or due date changes, these have not been considered in the SC context, which has been usually concerned with the behavioral correctness of the system actions.

2.5 Controller Implementation

As mentioned earlier in this chapter, a formal method for transforming a deadlock-free schedule into an implementable supervisor has not been reported in literature. Moreover, there is a structural difference between most of the abstract supervisors proposed in literature and the *implementable* one that can realize the obtained behavior. Most

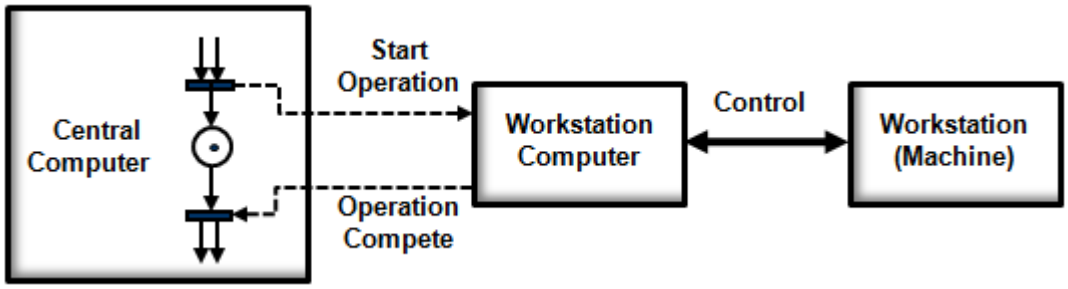
supervisor models, whether automata or PN, evolve asynchronously, and some PN models allow concurrency. Physical implementation devices, however, are synchronous and strictly sequential [100]. In other words, industrial controllers, like Programmable Logic Controllers (PLCs), do not have the power to choose from competing enabled events, or the power of resolving conflicts. This has imposed many challenging technical problems in the implementation stage.

In accordance with the work proposed in this study, focus in this section will be directed towards implementation approaches that utilized PN supervisor models. In general, there are two possible techniques that can be followed to implement a logic controller using a PN model; interpret the PN model at run time using an industrial computer, or transform the PN model using a compiler into a PLC program [101]. While the first requires a fast industrial computer to realize the control actions in real time, the second requires a substantial time to regenerate the PLC code every time a change occurs in the system. However, the first approach provides more flexibility, and enables defining generic rules that can be utilized to resolve conflicts or concurrency situations in the PN models.

In [33], a PN-based supervisory controller for a FMS was implemented through a hierarchical architecture that employed a central computer to supervise lower level computers that directly control the system devices (Figure 2.8). The PN graphical notation of the model was transformed into a text representation using a PN descriptive language (PNDL). Each text file defined information about places, transitions, and the connections between them. A PN supervisory system (PNSS) was then utilized to

interpret the information given by the PNDL and deliver it to the device controllers. In the case of concurrency among enabled transitions, priority policies like first-in-first-out (FIFO) were used to select the transition to be fired.

Figure 2.8: Computer-based hierarchical control



In [102], a user-friendly computer-based control system was developed using PN editors (model designers) and token players (simulators). The system consisted of three main parts; i) The PN editor and token player, ii) a transition mapping server whose primary function was to map the transitions of the token player to those in the equipment drivers, and iii) an equipment driver implemented for each piece of equipment within the controlled cell. Like the control system shown in Figure 2.8, in [102] the PN model represented an operation by a command transition that signify the start of the operation, followed by a place signifying the status of continuation of the operation, and a response transition signifying the completion of the operation. It was stated that the main advantage of this approach was enabling the user to modify the operation flow directly on the modeled PN without programming efforts.

The IEC 1131 standard defines four languages for programming PLCs; instruction lists, structured texts, logic ladder diagrams (LLDs), and function block diagrams. To

structure larger software projects, the sequential function chart (SFC) graphical language was also provided. Furthermore, to overcome structural conflict problems in PN models resulting from nondeterministic firing of transitions, Grafcet was developed and accepted as an international standard for designing PLC programs [103]. The supervisory control approaches that transformed PN models into PLC programs have utilized a variety of these languages. In [101] and [103], colored PNs (CPNs) were utilized to represent the supervisor. Actions to be transmitted to the controlled system were associated with the places of the CPN, and signals coming from the system were defined as input conditions to the transitions of the CPN. The CPN model was then transformed into a textual description that defines the places, transitions, and initial marking of the net. This textual description was finally compiled in a structured text to program the PLC.

In [104], SFC was utilized to implement the controller. Like PNs, SFCs are defined by two types of nodes, steps that resemble the places of a PN, and transitions. The main differences between PNs and SFCs are that SFCs can only handle one token in each step (place), and cannot represent transitions in conflict. To overcome these differences in [104], *safe* MGs were used to model the controller. MGs by definition do not feature conflicts, and safeness implies that each place can at most have one token in each reachable state of the net. In [15], automation PNs (APNs) (Section 2.2.3.1) were transformed into LLDs to implement the supervisor on a PLC. The places and transitions of the APN were directly transformed into the corresponding LLD by defining a separate rung for each transition and some places in the net. Like [101], sensor readings from the system were considered as inputs to the controller and used as firing conditions on their output transitions. Finally, in [100], a modular PN approach was proposed to develop a

supervisor for a batch process. In accordance with the modular concept, macro tasks that entailed a sequence of micro tasks to be carried out by the plant were defined. To implement the control logic, first these macro tasks were transformed into SFCs. Eventually, these SFCs and the other portions of the PN model were transformed into a LLD to program the PLC.

2.6 Conclusions

From the review of literature, it can be generally concluded that the literature lacks a formal approach that can transform a deadlock-free schedule of a job shop system into an implementable supervisor. As mentioned earlier, the existence of such an approach will guarantee, not only the correct behavior of the system, but an optimized one as well. Furthermore, most deadlock-free scheduling approaches, with the exceptions of a very few, rely on the available SC approaches for obtaining deadlock-free schedules. Indeed, most of the reviewed studies approached the scheduling problem by searching the feasible state space generated by solving the SC problem. However, the SC approaches, especially those following the automata formalism and the SCT, require considerable amount of time and resources to solve problems normally encountered in an industrial setting. Moreover, SC approaches do not feature real-time flexibility to changes when disturbances occur in the system, especially disturbances related to product mix changes (addition of new jobs to the system).

It can then be deduced that there is a need for integrated approaches that can ensure an optimized and correct performance for the job-shop production systems, and be able to react to sudden disturbances in real time. Furthermore, such an integrated approach should provide a readily implementable supervisor to realize such performance. Along with this general conclusion, the following points outline more specific observations that can be deduced from the review:

- The job shop deadlock-free scheduling problem has been hardly approached using mathematical models. This is due to the aforementioned hardness of representing the capacities of buffers in the models. Yet, solving a mathematical model of the problem using the available off-the-shelf software on today's fast computers can provide the optimal solutions for small problems in an acceptable time.
- Routing flexibility has not been addressed by the mathematical models that were proposed to solve the deadlock-free scheduling problem in job shops.
- Very few studies have considered the deadlock-free reactive scheduling problem. In addition, in most of these studies, deadlock consideration has been conducted from an avoidance perspective, rather than from a prevention one, which does not guarantee the realization of the optimized schedules upon implementation.
- Most of the literature that considered introducing a new job to the product mix has followed the job insertion scheduling approach rather than the total rescheduling (TR) approach. In addition, some studies have concluded that, in reactive scheduling in general, TR can take significantly longer time and can result in more system nervousness while providing a better schedule. However, it has not been conclusively

stated when it is more beneficial to apply TR and which factors may affect these conclusions.

- The reactive scheduling literature lacks a generic approach that can handle a wide variety of disruptions in real time, while providing deadlock-free schedules.
- PN supervisors embedding a MG structure can be easily verified for liveness by ensuring that all circuits are marked. Furthermore, these supervisors are more suitable for direct implementation since they do not feature any conflicts in their structure.
- PN supervisor models can be easily transformed into a variety of programming languages for PLCs. They can also be implemented through an industrial computer, which is directly connected to the system to be controlled. The latter approach provides more flexibility for changes and permits setting rules for resolving concurrency problems.

CHAPTER 3:

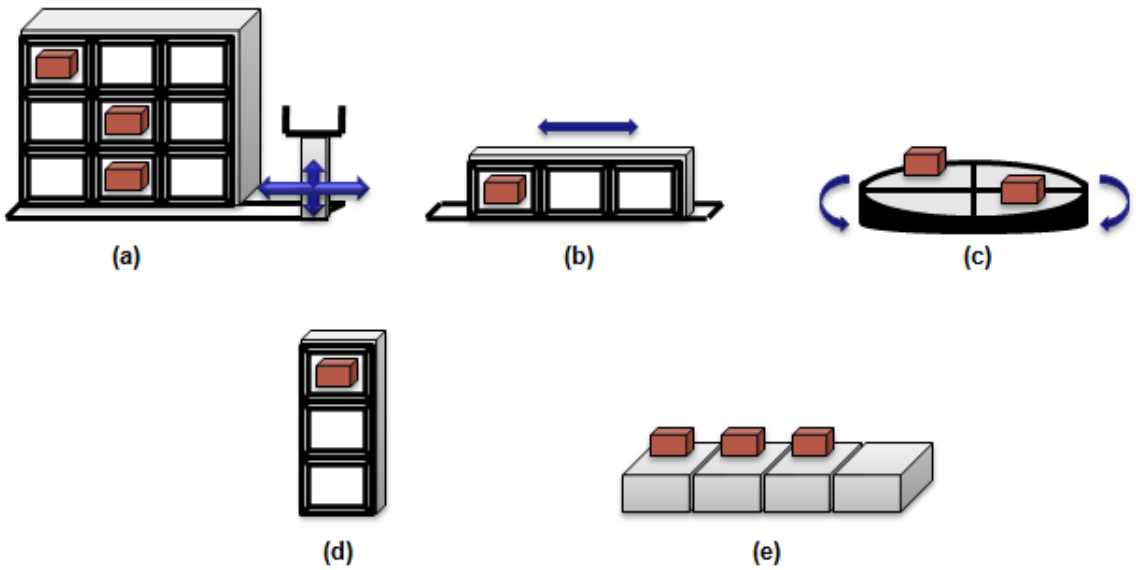
Deadlock-free Scheduling

3.1 Introduction

As indicated in Section 1.4, a scheduler should constitute the highest level in any efficient control architecture of a production system. The function of a lower level supervisor would then be to implement, in the shop floor, the decisions reached by the scheduler. In automated systems, scheduling decisions should ensure that the controlled system will operate in a correct manner, such that no deadlock situations are reached. Accordingly, the considered deadlock-free scheduling problem in automated flexible job shops can be summarized as follows: to optimize the schedule, based on a given criterion, of a number of jobs, where each has a number of operations to be performed on *some or all* of the available machines in the system (shop). In this thesis, the objectives considered are either, minimizing the mean flow time (MFT) or the makespan (MS). Some or all of the jobs may have alternative routes through the system. The system may have some buffer capacity that can be used in preventing blocking of machines or resolving circular waits. In addition, a transporter (e.g. robot) is used to perform all material handling operations between machines and other machines or buffers.

Different types of intermediate (machine input) and central buffers have been considered in industry and in the previous literature. These ranged from simple vertical or horizontal pallet stackers to Automated Storage and Retrieval Systems (ASRS) as shown in Figure 3.1. In [105], some existing manufacturing systems, which have the five types of buffer settings shown in the figure, were listed. Because a central buffer is usually served by the material handler serving the machines in the system, it usually features a simple, unsophisticated setting like the vertical or horizontal pallet stackers [105, 106, 107]. On the other hand, in the case of intermediate buffers (or input buffers to machines), either a secondary material handler is set between a simple pallet stacker and the corresponding machine, or a pallet indexing shuttle with a machine feeding mechanism is utilized [105].

Figure 3.1: Different types of buffers; (a) ASRS with "pallet mover" elevator; (b) linear pallet indexing shuttle; (c) rotary pallet indexing shuttle; (d) vertical pallet stacker; (e) horizontal (linear) pallet stacker



Whether intermediate or central, the choice of the buffer configuration (horizontal, vertical, or rotary) has merely depended on the type of material handling equipment and the available space in the system. In the current study, since focus is directed towards FMCs, which usually employ a robot arm as the primary material handler, central buffers can be assumed to be vertical pallet stackers, while intermediate buffers can either feature linear or rotary pallet indexing shuttles.

To solve the defined deadlock-free scheduling problem, two approaches are proposed in this chapter. The first approach (Section 3.2) utilizes Mixed-Integer-Programming (MIP) formulations, where deadlocks are prevented by ensuring that a job does not block any machine while waiting for its next resource to become available. To account for the different types of buffers that can be available in the system and how they function, four MIP models are proposed. Because of the aforementioned hardness of solving such models in a reasonable time for medium and large problems, a second approach is proposed (Section 3.3). This approach is an operations insertion algorithm (OI) that utilizes rank matrices to generate the schedules and prevent or resolve circular waits as well as unfeasible job sequences, taking into consideration the available buffer capacity. To obtain more realistic schedules, a transportation operations insertion algorithm (TOI) is also proposed (Section 3.4). This algorithm can be either used to insert the transportation operations in the schedules obtained using the MIP models or OI, or augmented in OI to consider the transportation operations in the schedule building process. In Section 3.5, a numerical example is solved using both approaches for illustration. In Section 3.6, the performances of the MIP models and OI are evaluated.

3.2 Mathematical Modeling

In [70], Ramaswamy and Joshi proposed MIP formulations to solve the deadlock-free scheduling problem of automated job shops for two different cases; in the first, no buffer space was present in the system (formulation RJP_1), and in the second, intermediate buffers were used to swap jobs between machines (formulation RJP_3). The objective in these models was to ensure that jobs did not block any machines, and thus the occurrence of deadlocks could be prevented. This was achieved in *formulation RJP_1* by guaranteeing that a job, after finishing processing on a machine, always finds the next machine in its processing route available as follows:

$$Minimize \quad Z = \sum_{i=1}^{n} x_{iK} \tag{3.1}$$

$$\begin{aligned} &subject\ to: \\ &x_{ikj} - T_{ikj} \geq x_{i(k-1)l}, \qquad \forall i \in I \end{aligned} \tag{3.2}$$

$$x_{ikj} - x_{prj} + M(1 - y_{ikprj}) \geq T_{ikj}, \quad \forall j \in J; (i,p) \in I \tag{3.3}$$

$$x_{prj} - x_{ikl} + My_{ikprj} \geq T_{prj}, \quad \forall j \in J; (i,p) \in I \tag{3.4}$$

$$x_{i(k-1)l} - x_{prj} + M(1 - y_{ikprj}) \geq E, \quad \forall j \in J; (i,p) \in I \tag{3.5}$$

$$x_{p(r-1)w} - x_{ikj} + My_{ikprj} \geq E, \quad \forall j \in J; (i,p) \in I \tag{3.6}$$

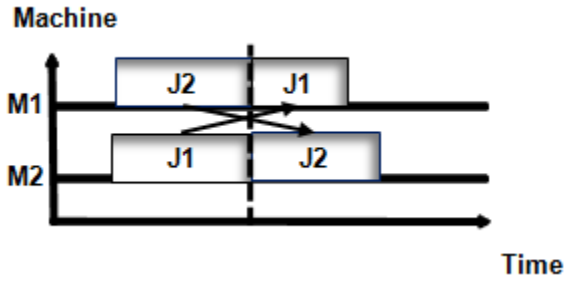
where I is the set of all jobs, J is the set of all machines, x_{iK} is the completion time of the last operation of job i, x_{ikj} is the completion time of operation k of job i on machine j, T_{ikj} is the processing time of operation k of job i on machine j, M is a large positive number,

E is a small positive number, and y_{ikprj} is equal to one if operation k of job i follows operation r of job p on machine j, and equal to zero otherwise.

In formulation RJP_1, constraint set (3.2) defined the processing time and routing information of each job. Constraint sets (3.3) and (3.4) restricted each machine to process not more than one job at a time. This was achieved by the binary variable y_{ikprj}, which, if equal to one, will force operation o_{ik} to follow o_{pr} on machine j or, if equal to zero, will force o_{pr} to follow o_{ik}. Constraint sets (3.5) and (3.6) were the ones that guaranteed the deadlock-freeness of the resulting schedules. This was achieved by ensuring that the operation completion time of a job on a machine must be greater than (at least by E) the completion time of any other job that precedes the first job on the next machine in its route. This guaranteed that a job would always find the next machine in its route available, and hence machine blocking, and consequently the occurrence of deadlocks was prevented.

Formulation RJP_1 provided deadlock-free schedules for job shop systems with no buffer space. Ramaswamy and Joshi [70] proposed *formulation RJP_3* to account for the presence of intermediate buffers between machines. The difference between formulations RJP_1 and RJP_3 was in setting the right-hand sides of constraint sets (3.5) and (3.6) to be equal to zero (instead of E) in formulation RJP_3. This modification allowed circular transfer of jobs between machines to resolve deadlocks. In other words, machines were allowed to *swap* the jobs (Figure 3.2) by using the intermediate buffer space.

Figure 3.2: Swapping of jobs between machines



In Figure 3.2, if no buffer space is present in the system, the circular wait between jobs J_1 and J_2 on machines M_1 and M_2 would force the system into a deadlock. If an intermediate buffer between M_1 and M_2 is available, J_1 (J_2) can reside in it until J_2 (J_1) acquires M_1 (M_2), after which J_1 (J_2) can acquire M_2 (M_1) as shown in Figure 3.2. Formulation RJP_3 allowed this kind of swapping by setting the right hand sides of constraint sets (3.5) and (3.6) to zero. However, solving formulation RJP_3 does not yield optimal schedules as mentioned in [70]. This is because the capacities of buffers were not represented in this formulation; alternatively, the virtual existence of these buffers was only implicit in the setting of constraint sets 3.5 and 3.6. Accordingly, jobs were only allowed to reside instantaneously in these buffers while being swapped and the capacities of these buffers were underutilized.

The RJP_1 model for the deadlock-free scheduling problem where no buffers existed was efficiently formulated and obtained optimal solutions for the problem. Accordingly, the objective of proposing the following formulations is to obtain schedules capable of efficiently utilizing the available buffer space in the system. In the proposed models,

deadlocks are avoided by preventing jobs from blocking machines. Four different models are proposed to take into account the configuration of buffer capacity:

- Intermediate buffers used only for swapping jobs (IBS).
- Intermediate buffers with unit capacity (IB1).
- Intermediate buffers with arbitrary capacity (IBA).
- Central buffer with arbitrary capacity (CBA).

The objective in all these models is to guarantee that a job, after finishing processing on a machine, always finds an available machine or a buffer space in its processing route. Hence, machine blocking, and consequently deadlocks, can be prevented. Note that the objective criterion considered in the following models is minimizing the MFT [70]. However, any objective criterion related to "completion times" such as minimizing the MS, can also be handled by these models.

3.2.1 Notation

The following notation is used in the proposed models:

Indices:

i, p, q	jobs (job types),
k, r	order of operations in processing routes of jobs, and
j, s, l, w, a	machines.

Sets:

I set of all n jobs

J set of all m machines

A_{ik} set of alternative machines that can process operation ik, and

P_j set of operations processed on machine j.

Parameters:

T_{ikj} processing time of job i operation k on machine j,

o_{ik} operation k of job i

O_i number of operations of job i

B_j capacity of intermediate buffer before (input buffer to) machine j,

B capacity of central buffer,

M very large positive number, and

E very small positive number.

Variables:

x_{ikj} completion time of job i operation k on machine j,

y_{ikprj} $= \begin{cases} 1 \text{ if } o_{ik} \text{ follows } o_{pr} \text{ on machine } j \\ 0 \text{ otherwise,} \end{cases}$

b_{ik} $= \begin{cases} 1 \text{ if job } i \text{ resides in a buffer before starting } o_{ik} \\ 0 \text{ otherwise,} \end{cases}$

b_{ikpr1} $= \begin{cases} 1 \text{ if } o_{ik} \text{ is started after } o_{p(r-1)} \text{ is completed} \\ 0 \text{ otherwise,} \end{cases}$ and

b_{ikpr2} $= \begin{cases} 1 \text{ if } o_{ik} \text{ is started before } o_{pr} \text{ is started} \\ 0 \text{ otherwise.} \end{cases}$

3.2.2 The IBS Model

In this model, it is assumed that the intermediate buffers are only used to swap jobs between machines (Figure 3.2), like the model earlier proposed in [70] for the case of intermediate buffers (RJP_3). The IBS model is formulated as follows:

$$Minimize \quad Z = \sum_{i=1}^{n} x_{ikj}, \quad k = O_i; j \in J \tag{3.7}$$

$$\begin{aligned} &subject\ to: \\ &x_{ikj} \geq T_{ikj}, \qquad \forall i \in I; j \in J; k = 1 \end{aligned} \tag{3.8}$$

$$x_{ikj} - T_{ikj} \geq x_{i(k-1)l}, \qquad i \in I; 1 < k \leq O_i; j, l \in J \tag{3.9}$$

$$x_{i(k-1)l} - x_{prj} + M(1 - y_{ikprj}) \geq 0, \quad \forall j \in J; i, p \in I; l \in J; k > 1; o_{ik}, o_{pr} \in P_j \tag{3.10}$$

$$x_{prj} - x_{i(k-1)l} + My_{ikprj} \geq 0, \quad \forall j \in J; i, p \in I; l \in J; r > 1; o_{ik}, o_{pr} \in P_j \tag{3.11}$$

$$x_{p(r-1)w} - x_{ikj} + My_{ikprj} \geq 0, \quad \forall j \in J; i, p \in I; w \in J; r > 1; o_{ik}, o_{pr} \in P_j \tag{3.12}$$

$$x_{ikj} - x_{prj} + M(1 - y_{ikprj}) \geq T_{ikj}, \quad \forall j \in J; i, p \in I; k, r = 1; o_{ik}, o_{pr} \in P_j \tag{3.13}$$

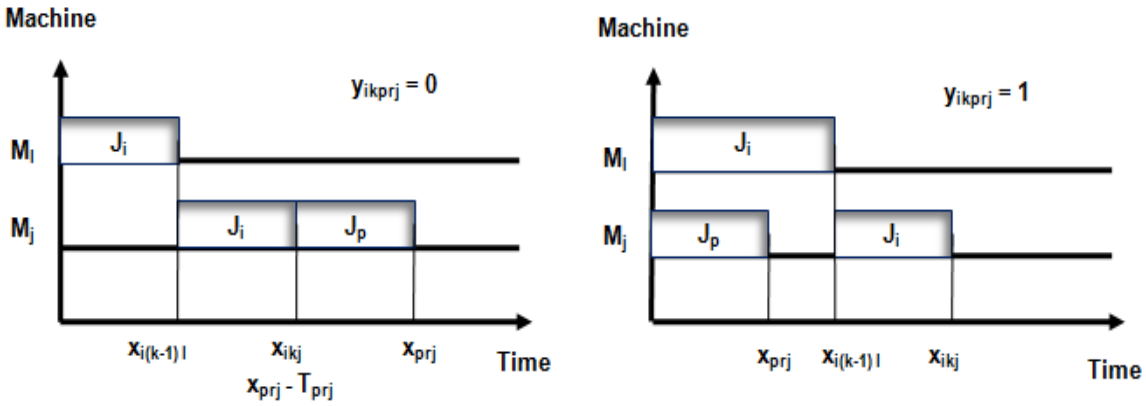
$$x_{prj} - x_{ikj} + My_{ikprj} \geq T_{prj}, \quad \forall j \in J; i, p \in I; k, r = 1; o_{ik}, o_{pr} \in P_j \tag{3.14}$$

$$y_{ikprj} \in \{0,1\} \quad \forall j \in J; i, p \in I; o_{ik}, o_{pr} \in P_j \tag{3.15}$$

In the above model, constraint sets (3.8) and (3.9) define the precedence relations between operations in the processing route of each job. Provided that o_{ik} is processed on machine j, constraint sets (3.10) and (3.11) force the binary variable y_{ikprj} to be equal to one if the finishing time of $o_{i(k-1)}$ is greater than or equal to the finishing time of o_{pr} on machine j, and zero otherwise. If y_{ikprj} is equal to zero, constraint set (3.12) will ensure that o_{pr} is processed after o_{ik} on machine j. The combined effect of constraint sets (3.10),

(3.11), and (3.12) ensures that, processing of a job i on a machine j precedes the processing of any other job p on this machine if job p is completed on machine j after job i is completed on the previous machine in its route. This guarantees that when a job i is completed on a machine l, the successor machine j in its route will be available, and hence blocking of machines can be prevented as shown in Figure 3.3.

Figure 3.3: Effect of y_{ikprj} in the IBS model



Constraint sets (3.10), (3.11), and (3.12) along with constraint set (3.9) will further ensure that no machine is simultaneously occupied by more than one job at any time. However, since these constraint sets are defined for operations that have predecessors in their jobs' routes, constraint sets (3.13) and (3.14) are added to maintain the same restriction only between the first operations of jobs that share the same machine. In comparison to the RJP_3 model, the IBS model only uses three constraint sets (3.10, 3.11, and 3.12) to mutually ensure that, no machine is simultaneously occupied by more than one job at any time and a job will always find the next machine in its route available. In the RJP_3 model, however, four constraint sets were used to achieve these two goals. Note that the number of constraints in sets (3.10), (3.11), and (3.12) are all equal, and that the

RJP_3 model has the same number of constraints in the corresponding sets. Furthermore, the number of additional constraint sets, (3.13) and (3.14), in the proposed IBS model is limited because these constraints are defined only between the first operations of jobs that share the same machine. Hence, the number of constraints in the IBS model is potentially less than that in the RJP_3 model. Accordingly, the IBS model can be solved to obtain the same objective values that can be obtained by the RJP_3 model in potentially less computing time.

3.2.3 The IB1 Model

In IBS and RJP_3 models, the capacities of the intermediate buffers are not represented in the model. Accordingly, solving these models to optimality will not guarantee the optimal performance of the system, since the buffers will only be utilized to swap jobs. In the following model this problem is overcome, where a job can reside in the intermediate buffer while waiting for the corresponding machine to become available. However, it is assumed that the buffers are unit capacity buffers, and that a job residing in a buffer must precede any other job on the corresponding machine when this machine becomes available. In other words, a machine feeding mechanism is assumed to automatically feed the machine with the job in the buffer as soon as the machine becomes available. Machine feeding mechanisms are routinely used in automated systems.

This model is similar to RJP_3 except for the modifications in constraint sets (3.21) and (3.22) as follows:

$$Minimize \quad Z = \sum_{i=1}^{n} x_{ikj}, \quad k = O_i ; j \in J \tag{3.16}$$

$$\begin{aligned} &subject\ to: \\ &x_{ikj} \geq T_{ikj}, \qquad \forall i \in I; j \in J; k = 1 \end{aligned} \tag{3.17}$$

$$x_{ikj} - T_{ikj} \geq x_{i(k-1)l}, \qquad i \in I; 1 < k \leq O_i; j, l \in J \tag{3.18}$$

$$x_{ikj} - x_{prj} + M(1 - y_{ikprj}) \geq T_{ikj}, \quad \forall j \in J; i, p \in I; l \in J; o_{ik}, o_{pr} \in P_j \tag{3.19}$$

$$x_{prj} - x_{ikl} + My_{ikprj} \geq T_{prj}, \quad \forall j \in J; i, p \in I; l \in J; o_{ik}, o_{pr} \in P_j \tag{3.20}$$

$$\begin{aligned} &x_{i(k-1)l} - (x_{prj} - T_{prj}) + M(1 - y_{ikprj}) \geq 0, \\ &\forall j \in J; i, p \in I; k > 1; l \in J; o_{ik}, o_{pr} \in P_j \end{aligned} \tag{3.21}$$

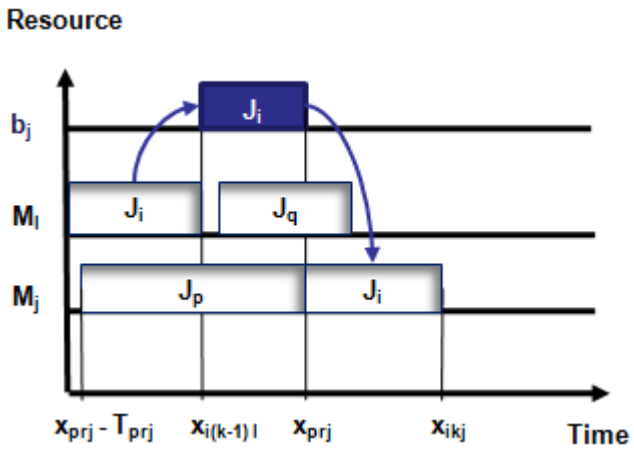
$$x_{p(r-1)w} - (x_{ikj} - T_{ikj}) + My_{ikprj} \geq 0, \quad \forall j \in J; i, p \in I; r > 1; w \in J; o_{ik}, o_{pr} \in P_j \tag{3.22}$$

$$y_{ikprj} \in \{0,1\} \quad \forall j \in J; i, p \in I; o_{ik}, o_{pr} \in P_j \tag{3.23}$$

Similar to the RJP_1 model, constraint sets (3.17) and (3.18) conserve the processing routes of the jobs. Constraint sets (3.19) and (3.20) ensure that a machine will not be simultaneously occupied by two jobs. If y_{ikprj} is equal to one, constraint set (3.21) will ensure that job i is completed on the previous machine in its route, l, with or after job p is *started* on machine j, while constraint set (3.22) will ensure the opposite, otherwise. The difference between these constraint sets and the ones in RJP_3 is adding the processing time terms T_{prj} and T_{ikj} to the left hand sides of constraint sets (3.21) and (3.22), respectively. The effect of this modification is shown in Figure 3.4. In this figure it can be seen that, adding the processing time term in constraint set (3.21) enables the completion of job i on machine l as soon as job p is started on machine j. It can then reside in the intermediate buffer b_j before machine j until machine j becomes available,

thus making machine *l* available to process any other job *q*. Without this modification, as in IBS, RJP_1 or RJP_3, job *i* can only be completed on machine *l* after or when job *p is* completed on machine *j*.

Figure 3.4: Utilizing capacity of intermediate buffer



3.2.4 The IBA Model

In the IB1 model, only one job can reside in an intermediate buffer at any time. Furthermore, this job has to be the first job to be processed on the corresponding machine when it becomes available. In the IBA model, however, any number of jobs is allowed to simultaneously utilize the available capacity of the intermediate buffers. There is also no restriction on the order by which these jobs are dispatched from the buffer. Thus, the obtained schedule will determine this dispatching order. This gives more flexibility to the cell, but at the expense of more computational time required to solve the model; representing the capacities of the buffers in the model require the use of additional binary variables. The IBA model is formulated as follows:

$$Minimize \quad Z = \sum_{i=1}^{n} x_{ikj}, \quad k = O_i; j \in J \tag{3.24}$$

$$\begin{aligned} &subject\ to: \\ &x_{ikj} \geq T_{ikj}, \qquad \forall i \in I; j \in J; k = 1 \end{aligned} \tag{3.25}$$

$$x_{ikj} - T_{ikj} \geq x_{i(k-1)l}, \qquad i \in I; 1 < k \leq O_i; j,l \in J \tag{3.26}$$

$$x_{ikj} - x_{prj} + M(1 - y_{ikprj}) \geq T_{ikj}, \quad \forall j \in J; i,p \in I; l \in J; k > 1; o_{ik}, o_{pr} \in P_j \tag{3.27}$$

$$x_{prj} - x_{ikl} + My_{ikprj} \geq T_{prj}, \quad \forall j \in J; i,p \in I; l \in J; k > 1; o_{ik}, o_{pr} \in P_j \tag{3.28}$$

$$(x_{ikj} - T_{ikj}) - x_{i(k-1)l} - Mb_{ik} \leq 0 \qquad \forall i \in I; 1 < k \leq O_i; j,l \in J \tag{3.29}$$

$$(x_{ikj} - T_{ikj}) - x_{p(r-1)w} - Mb_{ikpr1} \leq 0, \quad j,w \in J; i,p \in I; k,r > 1; o_{ik}, o_{pr} \in P_j \tag{3.30}$$

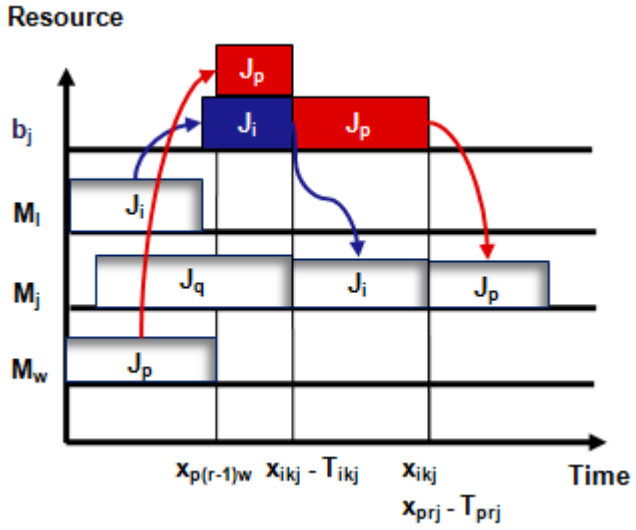
$$\begin{aligned} &b_{ik} + \sum_{p \neq i} [(1 - y_{ikprj}) + b_{ikpr1} - 1] \leq B_j, \\ &\forall i \in I; 1 < k \leq O_i; p \in I; j \in J; o_{ik}, o_{pr} \in P_j \end{aligned} \tag{3.31}$$

$$\begin{aligned} &y_{ikprj} \in \{0,1\} \quad \forall j \in J; i,p \in I; o_{ik}, o_{pr} \in P_j \\ &b_{ik} \in \{0,1\} \quad \forall i \in I; 1 < k \leq O_i \\ &b_{ikpr1} \in \{0,1\} \quad i,p \in I; j \in J; k,r > 1; o_{ik}, o_{pr} \in P_j \end{aligned} \tag{3.32}$$

Constraint sets (3.25), (3.26), (3.27), and (3.28) are the same as before, where the first two capture the processing routes of the jobs, and the second two prevent a machine from being simultaneously occupied by more than one job. In constraint set (3.29) the binary variable b_{ik} will be equal to one if the starting time of o_{ik} on machine j is greater than the finishing time of $o_{i(k-1)}$ on machine l. This indicates that o_{ik} will not be started on machine j immediately after operation $o_{i(k-1)}$ is completed; hence job i will reside in the buffer preceding machine j until it becomes available. If o_{pr} is processed on machine j, constraint set (3.30) will force the binary variable b_{ikpr1} to be equal to one if o_{ik} is started

on machine j after $o_{p(r-1)}$ is completed on some machine w. As shown in Figure 3.5, having b_{ik} and b_{ikpr1} equal to one and y_{ikprj} equal to zero, indicates that for a period of time jobs i and p will share the capacity of the intermediate buffer b_j.

Figure 3.5: Utilizing an intermediate buffer with arbitrary capacity



Finally, constraint set (3.31) ensures that if any job visits the intermediate buffer preceding a machine, the total number of jobs residing in that buffer at that time will not exceed its capacity. As mentioned earlier, having $o_{ik} \in P_j$ and b_{ik} equal to one indicate that job i will visit b_j. Furthermore, the term $[(1 - y_{ikprj}) + b_{ikpr1} - 1]$ will be equal to one iff a job p is residing in this buffer when job i is started on machine j. Otherwise, this term will be equal to zero as shown in Table 3.1. Summing the term $[(1 - y_{ikprj}) + b_{ikpr1} - 1]$ for all jobs p and adding the result to b_{ik} will give the total number of jobs residing in the intermediate buffer when job i resides in it.

Table 3.1: Different combinations of the values of the binary variables

y_{ikprj}	b_{ikpr1}	$[(1 - y_{ikprj}) + b_{ikpr1} - 1]$	Indication
1	**0**	**-1**	*Not applicable* since $x_{p(r-1)w}$ cannot be greater than $x_{prj} - T_{prj}$
1	**1**	**0**	o_{ik} is started on machine j after o_{pr} has already been started on machine j
0	**0**	**0**	$o_{p(r-1)}$ is finished on machine w after o_{ik} has already been started on machine j
0	**1**	**1**	o_{ik} is started on machine j after $o_{p(r-1)}$ is finished on machine w, but before o_{pr} is started on machine j (Figure 3.5)

3.2.5 The CBA Model

Unlike the intermediate buffers, a central buffer, not allocated to any specific machine, can serve the whole cell (system). This type of buffers can sometimes be more suitable in cells featuring a limited space problem, where there is no adequate space to attach a buffer to every machine. Furthermore, one central buffer with a capacity equal to B can sometimes perform as well as a number of intermediate buffers equal to the number of machines, each with a capacity equal to B. However, this will not be true if the capacities of the intermediate buffers are utilized simultaneously.

In the IBA model, since an intermediate buffer is associated with each machine, the number of jobs residing in a buffer was determined in part by the value of y_{ikprj} (constraint set (3.31)). However, this cannot be applied in the CBA model because the central buffer is not associated with any machine and is set to serve the whole cell. Accordingly, an additional group of binary variables, b_{ikpr2}, and constraints are added to the model to fully define the number of jobs residing in the buffer when a job i resides in

it. This will however result in more computational time to solve the CBA model than the previous models. The first five constraint sets in the CBA model are the same as those in the IBA model. The rest of the constraint sets are formulated as follows:

$$(x_{ikj} - T_{ikj}) - x_{p(r-1)w} - Mb_{ikpr1} \leq 0, \quad \forall i, p \in I;\ j, w \in J;\ k, r > 1 \tag{3.33}$$

$$(x_{prs} - T_{prs}) - (x_{ikj} - T_{ikj}) - Mb_{ikpr2} \leq 0, \quad \forall i, p \in I;\ j, s \in J;\ k, r > 1 \tag{3.34}$$

$$b_{ik} + \sum_{p \neq i} [b_{ikpr1} + b_{ikpr2} - 1] \leq B, \quad \forall i \in I; 1 < k \leq O_i;\ p \in I;\ k, r > 1 \tag{3.35}$$

$$\begin{aligned} &y_{ikprj} \in \{0,1\} \quad \forall j \in J;\ i, p \in I;\ o_{ik}, o_{pr} \in P_j \\ &b_{ik} \in \{0,1\} \quad \forall i \in I; 1 < k \leq O_i \\ &b_{ikpr1}, b_{ikpr2} \in \{0,1\} \quad \forall i, p \in I;\ k, r > 1 \end{aligned} \tag{3.36}$$

In the above constraint sets, b_{ikpr2} performs the same function of y_{ikprj} in constraint set (3.31) of the IBA model. However, note that unlike the IBA model, in constraint sets (3.33), (3.34), and (3.35), o_{ik} and o_{pr} do not have to be processed on the same machine. Furthermore, the combinations (0,0), (0,1), (1,0), and (1,1) of binary variables b_{ikpr2} and b_{ikpr1} will have the same indications shown in Table 3.1 for the combinations (1,0), (1,1), (0,0), and (0,1), respectively, of y_{ikprj} and b_{ikpr1}. Hence, a job p will be in the central buffer when another job i is started on a machine j iff b_{ikpr2} and b_{ikpr1} are both equal to one.

It should be noted that in the IBA and the CBA models, it is assumed that swapping devices attached to machines [108] are used if instantaneous swapping of jobs (Figure 3.2) between machines would resolve a circular wait. This is because, in such a situation, there will be no difference between (x_{ikj} - T_{ikj}) and $x_{i(k-1)l}$; hence constraint set (3.29) will

not force the value of b_{ik} to be one, and consequently the models will not realize the need of any of the jobs involved in the circular wait to reside in the buffer.

3.2.6 Routing Flexibility

In the proposed mathematical models, routing flexibility will be defined as the possibility of processing an operation of a job on one of a set of alternative machines [68] (Note that a different definition for routing flexibility will be given in Section 3.3). Accordingly, the set of alternative machines for an operation o_{ik} is defined as $A_{ik,}$ first. Next, a binary variable y_{ika} is defined for each alternative as follows:

$$y_{ika} = \begin{cases} 1 \text{ if } o_{ik} \text{ is processed on machine } a, \ \forall\ a \in A_{ik} \\ 0 \text{ otherwise,} \end{cases}$$

To ensure that only one of the machines will be selected to process o_{ik}, the finishing times of this operation on the other machines in A_{ik} have to be equal to 0. Thus, these possible processes on unselected machines would start and end at time 0. This can be achieved by the following constraints:

$$x_{ika} \le M y_{ika}, \quad \forall\ a \in A_{ik} \tag{3.37}$$

$$\sum_{a \in A_{ik}} y_{ika} = 1, \quad i \in I; k < O_i \tag{3.38}$$

The original constraints of the model(s) can then be adapted by repeating the constraints that include o_{ik} each for $|A_{ik}|$ times, each associated with one of the machines

in A_{ik}. Finally, to eliminate the effects of the constraints involving the unselected machines, the term $M(1 - y_{ika})$ can be added to the greater-than side of these constraints. Thus when y_{ika} is equal to zero, indicating that machine a is not selected for o_{ik}, the constraints involving processing o_{ik} on machine a will become redundant.

3.3 Operations Insertion Algorithm

The MIP models proposed in the Section 3.2, along with the RJP_1 model provide optimal solutions for the cases and assumptions considered. However, the problem size that can be solved using these models, especially the central buffer model (CBA), is still limited. Furthermore, if transportation operations were to be included in these models, the complexity of the problem will grow dramatically [70]. As a result, in this section, an operations insertion algorithm is proposed. As will be shown, the algorithm can solve the deadlock-free scheduling problem when there is no buffer space, and when there is a central buffer with a limited capacity. It can account for different buffer sizes, transportation operations, and jobs with alternative routes, while providing deadlock-free near optimal (or optimal) schedules.

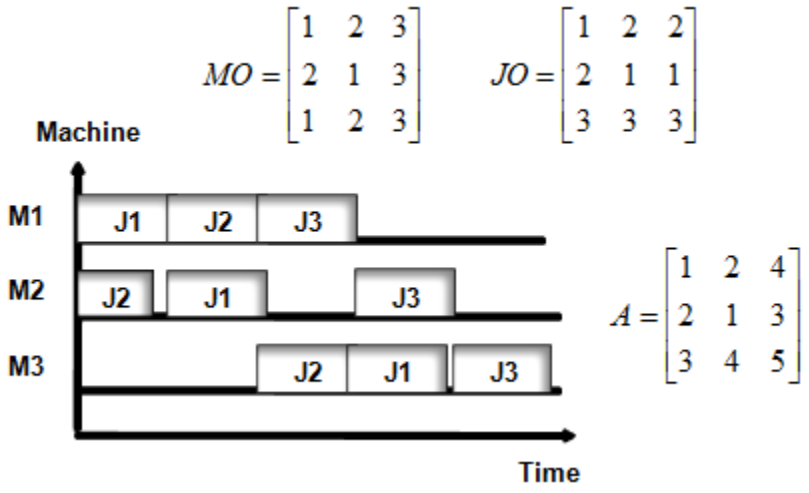
The proposed algorithm is a greedy one, because it is based on inserting the available jobs in the schedule one after the other, until a schedule for all the jobs is obtained. The generation of the schedules and the identification of deadlocks are based on the analysis of the associated rank matrices. In a rank matrix A of a schedule, each element p_{ij} corresponds to the maximal number of vertices from a source to vertex (i, j) of the

digraph representing the schedule. When each vertex (*i, j*) is assigned to an operation of a job *i* on a machine *j*, each row of *A* then represents the processing route of a job, and each column provides the sequence of jobs visiting each machine [75]. This rank matrix is equivalent to the special Latin rectangle (*LR*) that was proposed in [76]. In that study, a block-matrices-model was utilized in an insertion algorithm to solve the traditional job shop scheduling problem. The rows of a matrix *MO* represented the order of visiting machines in the processing route of each job. Knowing *MO*, iterations involving matrix *LR* were conducted to obtain a matrix *JO*. The columns of matrix *JO* provided the final order of each job in visiting each machine.

To illustrate, consider Figure 3.6 that shows the Gantt chart of a schedule of three jobs on three machines, matrix *MO* of the jobs, the *JO* matrix, and the corresponding rank matrix *A*. From this figure, it can be noticed that the rows of *A* are equivalent to the rows of *MO*, and that the columns of *A* are equivalent to the columns of *JO*. For example, the first row in *A*, like the first row in *MO*, indicates that job J_1 visits machines M_1, M_2, and then M_3 because the second element in that row in *A* is larger than the first and the third is larger than the second. Likewise, the second column in *A*, like the second column in *JO*, indicates that M_2 is visited by J_2, J_1 and then J_3.

Accordingly, in the proposed insertion algorithm, every time a new operation is inserted in the schedule, the rank matrix has to be updated to reflect its order on the corresponding machine. The rank p_{ij} of any operation that follows the newly inserted operation on that machine has to be incremented by one, along with the ranks of all the operations that follow it in job *i*'s route on the other machines (if any), and so on.

Figure 3.6: Rank matrix illustration

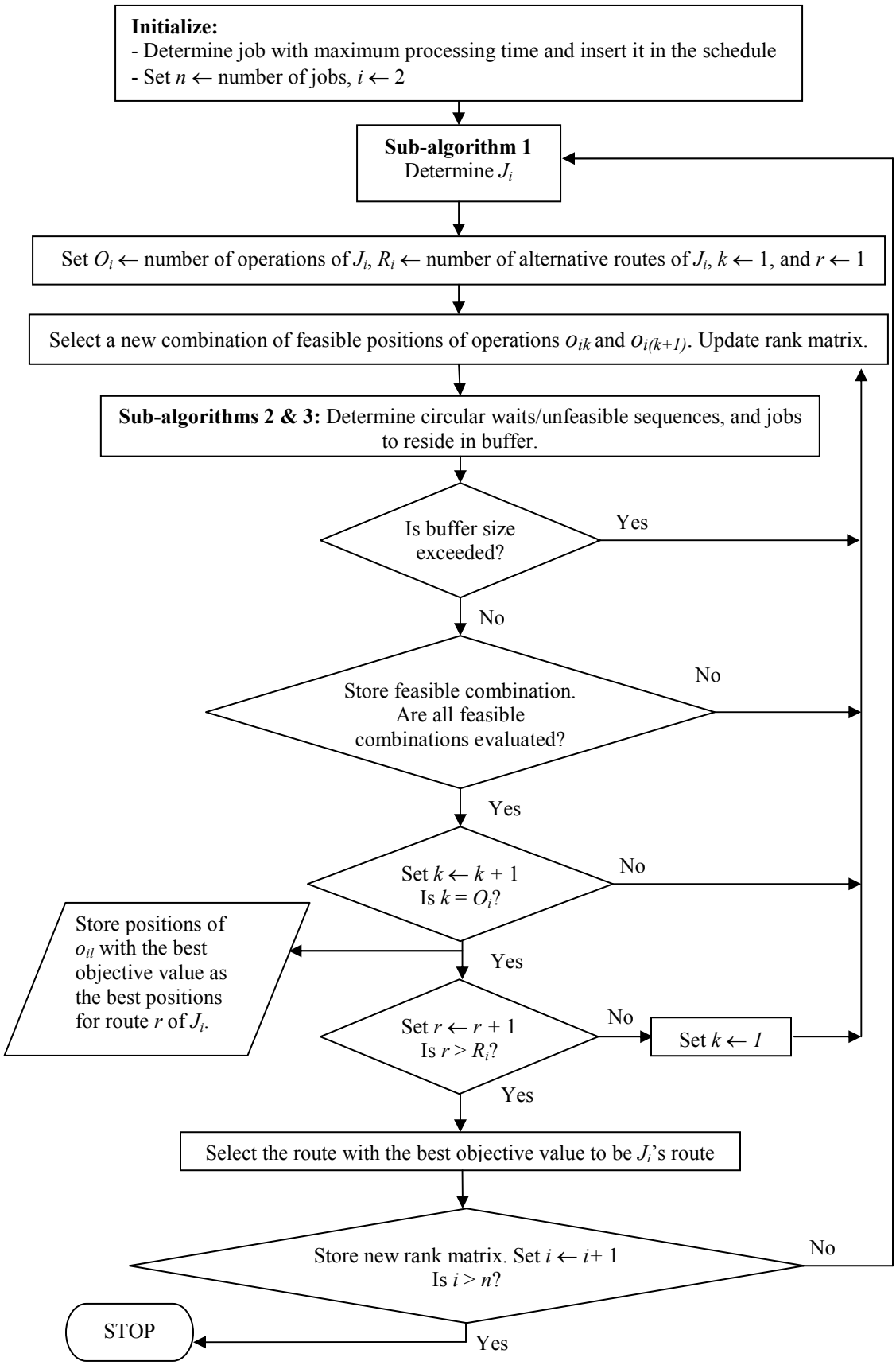


3.3.1 The Operations Insertion Algorithm (OI)

In the proposed algorithm, when operations are inserted, possible circular waits are detected from the rank matrix and are prevented or resolved according to the available buffer space. This ensures the deadlock-freeness of the resulting schedules. It should also be noted that in the proposed algorithm, routing flexibility is defined by the number of alternative routes that a job i can take through the system (R_i). This is a common measure of flexibility that has been considered in previous literature [109].

The following steps outline the proposed algorithm. Details of some of the steps are explained in the indicated sub-algorithms and sections. Figure 3.7 provides the flow chart of the algorithm.

Figure 3.7: Flowchart of Main Insertion Algorithm



Main Algorithm: Insertion algorithm

Step 1. Set $n \leftarrow$ number of jobs. Determine the job with the maximum processing time through the system and insert it in the schedule. Set $i \leftarrow 2$.

Step 2. Determine job J_i to be inserted (Sub-algorithm 1). Set $O_i \leftarrow$ number of operations of J_i, $R_i \leftarrow$ number of alternative routes of J_i, $k \leftarrow 1$, and $r \leftarrow 1$.

Step 3. For each combination of feasible positions of operation o_{ik} and all positions of $o_{i(k+1)}$ on the corresponding machines, do the following:

Step 3.1. Update the rank matrix.

Step 3.2. Check if a circular wait (Sub-algorithm 2) or an unfeasible sequence (Sub-algorithm 3) is formed. If a circular wait is formed, and none of the jobs in it is already scheduled to reside in the buffer, schedule J_i to reside in the buffer after completing o_{ik}.

Step 3.3. Estimate the starting and finishing times of all the operations, and the entrance and exit times of jobs scheduled to reside in the buffer. Compare the buffer size required to satisfy the current schedule with the buffer size available. If available size (zero or more) is exceeded, discard that combination and move to the next. Else, store the current combination as a feasible one along with its associated objective value (Section 3.3.2).

Step 3.4. (Not mandatory) Insert the transportation operations (Section 3.4).

Step 4. Store all feasible positions of $o_{i(k+1)}$, each along with the positions of o_{ih}, where $h = 1..k$, that result in the best objective value. Set $k \leftarrow k + 1$. If $k = O_i$, store the

positions of o_{ih} (where $h = 1..O_i$), that result in the best objective value, as the best positions for route r of J_i. Else, go to Step 3.

Step 5. Set $r \leftarrow r + 1$. If $r > R_i$, select the route with the best objective value to be J_i's route along with the associated positions of operations on the machines. Else, set $k \leftarrow 1$ and go to Step 3.

Step 6. Store the new rank matrix, and the new schedule.

Step 7. Set $i \leftarrow i + 1$. If $i > n$, Stop. Else, go to Step 2.

Sub-algorithm 1: Determining the job J_i to be inserted

Step 1. Calculate the flow factor f_u for each of the remaining un-inserted jobs as follows: each time a transition (e.g. M_j to M_l) in the processing route of an un-inserted job J_u appears in any of the processing routes of the jobs already inserted in the schedule, indicating a common move in the system, increment f_u by one.

Step 2. Select the job with the maximum flow factor, max(f_u), to be J_i. In case of ties, select the job with the maximum processing time through the system.

Sub-algorithm 2: Detecting Circular Waits

Step 1. Determine positions p_{ij} and p_{il} in the updated rank matrix, which correspond to operations o_{ik} and $o_{i(k+1)}$ respectively.

Step 2. From p_{il}, move vertically to each position p_{ql} with a smaller digit representing a job q that precedes job i on machine l.

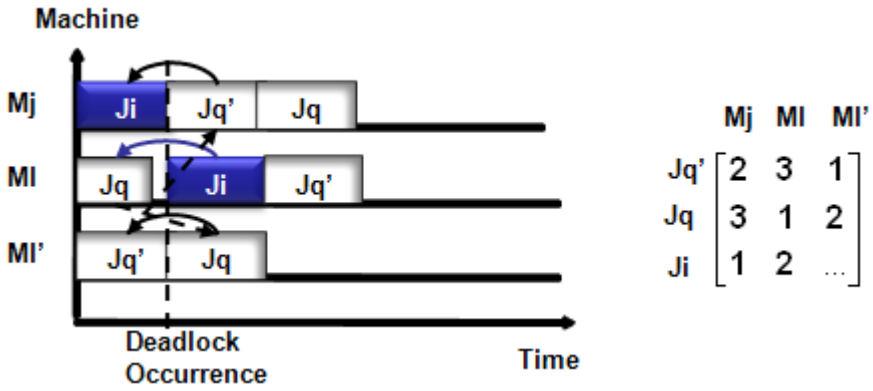
Step 3. From each p_{ql}, move horizontally and then vertically to determine the next machine in job q's route and the job preceding it on that machine.

Step 4. Repeat the cycle in Step 3 until one of the following conditions is met:

- If no position having a larger digit when moving horizontally or no position having a smaller digit when moving vertically is found, then there is no circular wait formed because of the considered positions of o_{ik} and $o_{i(k+1)}$.

- When moving vertically, if position p_{ij} is reached, this indicates the formation of a circular wait (Figure 3.8) because of the considered positions of o_{ik} and $o_{i(k+1)}$.

Figure 3.8: Detection of circular waits in a rank matrix

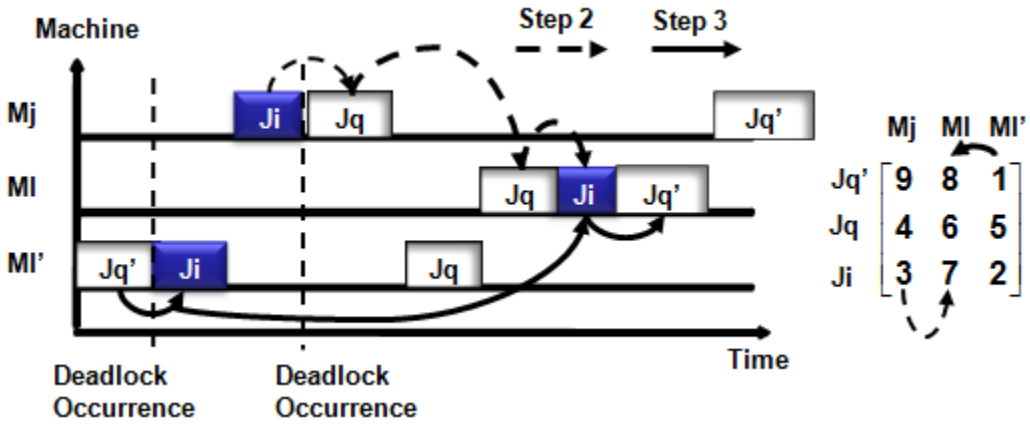


Sub-algorithm 3: Detecting Unfeasible Sequences (Figure 3.9)

Step 1. In the updated rank matrix, determine the pairs of columns associated with position p_{il} (that corresponds to $o_{i(k+1)}$) and each position $p_{il'}$ corresponding to a previously inserted operation of job i (including p_{ij} which corresponds to o_{ik}).

Step 2. If there are two consecutive positions in some row q, where the smaller is in column j and has a larger digit than that of p_{ij}, and the other is in column l and has a smaller digit than that of p_{il}, an unfeasible sequence is formed. Job i has to reside in the buffer after completion on machine j, otherwise a deadlock will occur.

Figure 3.9: Detection of unfeasible sequences in a rank matrix



Step 3. If there are two *directly* consecutive positions in some row *q'*, where the smaller is in column *l'* and has a smaller digit than that of $p_{il'}$, and the other is in column *l* and has a larger digit than that of p_{il}, an unfeasible sequence is formed. Job *q'* has to reside in the buffer after completion on machine *l'*.

3.3.2 Order of Jobs and Positions Evaluation

Due to the nature of the job shop problem, jobs often have different processing routes through the shop. Some of these routes might also be opposite in the flow direction. Such flow conflicts can cause deadlocks unless the available buffer space (if any) is utilized to store some jobs to allow others with conflicting routes to move safely through the system. Accordingly, in Step 1 of Sub-algorithm 1, a flow factor f_u is calculated for each un-inserted job, and the one with max(f_u) is selected for insertion. The objective behind that is to defer any potential conflicts in the schedule until a considerable number of jobs have

been inserted, at which stage the buffer space can be better utilized to acquire schedules with better objective values.

In Step 1 of the Main Algorithm, the job with the maximum processing time through the system is first selected for insertion. Furthermore, in Step 2 of Sub-algorithm 1, the maximum processing time through the system is used as a tie breaker between jobs sharing max(f_u). The objectives behind that are the following:

- Early consideration of operations featuring long processing times, provide better objective value estimates when evaluating the insertion positions of the operations to follow.
- Early consideration of jobs with long processing times increases the possibility for these jobs to find some available buffer space to reside in. This will permit other jobs, with operations having shorter processing times, to be completed earlier and thus will result in a better system mean flow time.

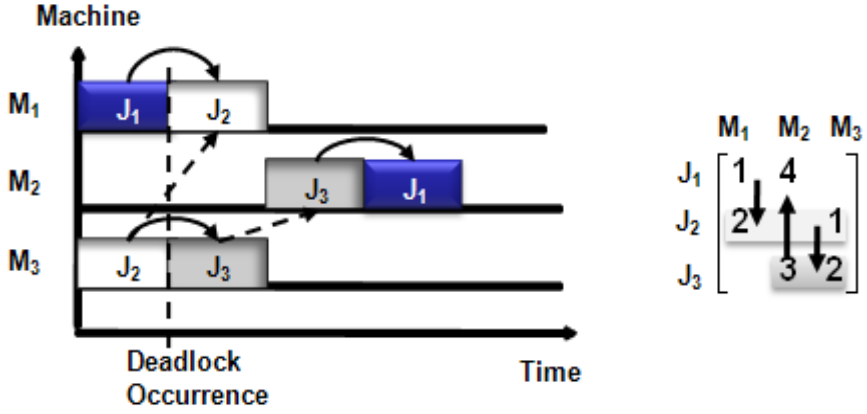
As for the evaluation of the positions of operations, it is mentioned in Step 3.3 of the Main Algorithm that, if the current combination of positions of operations o_{ik} and $o_{i(k+1)}$ on the machines is feasible, it is stored with its associated objective value (whether the objective is minimizing MS or MFT). Since at this stage usually other jobs are not yet inserted in the schedule, this associated objective value is estimated only based on the already inserted jobs. In addition, the finishing time of J_i in the system is approximated by adding the finishing time of operation $o_{i(k+1)}$ to the sum of processing times of the operations following $o_{i(k+1)}$ in J_i's route (if any).

3.3.3 Sufficiency for Deadlock Occurrence

The procedure outlined in sub-algorithm 2 aims at detecting any circular waits in the schedule as defined in the control context, where a set of jobs are each holding a resource while waiting for another held resource to become available (Figure 3.8). However, in the scheduling context the order of jobs on the machines is determined beforehand (before implementation on the shop floor), hence a job can be involved in a circular wait even before acquiring a resource (J_q on M_l and J_i on $M_{l'}$ in Figure 3.9). Since these situations do not exactly follow the definition of a circular wait (Section 1.1.2), they have been called unfeasible sequences in the current study. Sub-algorithm 3 is proposed to detect such sequences.

The presence of circular waits or unfeasible sequences, as detected by sub-algorithms 2 and 3, must result in a deadlock in the corresponding schedule. However, the application of these two sub-algorithms *separately* is not *sufficient* to detect all the deadlocks in a schedule. For example, consider Figure 3.10 that shows the schedule of three jobs on three machines and its associated rank matrix:

Figure 3.10: Complex circular waits



The deadlock situation shown in Figure 3.10 cannot be detected using any of sub-algorithms 2 or 3 on its own. Accordingly, in OI, these two sub-algorithms are combined to detect a more general circular wait condition that takes into account the unfeasible sequences shown above. The definition of this generalized condition and its sufficiency for the detection of deadlocks will be provided in Section 5.3.

3.3.4 Complexity of Algorithm

The worst case complexity of OI would be encountered if each job has to be processed on all the available m machines, and the capacity of the buffer space is large enough to permit the feasibility of all positions of operations o_{ik} and $o_{i(k+1)}$. In this case, assuming that $n-1$ jobs have already been inserted in the schedule, inserting the n^{th} job in the schedule would involve evaluating n positions for each of the operations o_{ik} and $o_{i(k+1)}$. This evaluation process will be repeated $(m-1)$ times to insert all the operations of the n^{th} job. Furthermore, within each evaluation, detecting circular waits and unfeasible sequences (sub-algorithms 2 and 3) involves $(m-1)(n-1)$ steps, equal to the number of pairs of directly consecutive operations of the previously inserted jobs on the corresponding machines. This results in a worst case complexity of $O(n^3m^2)$ for inserting the n^{th} job. More details on the complexity of detecting deadlocks while inserting an operation will be provided in Section 5.4.1.

3.4 Insertion of Transportation Operations

When the material handling (transportation) operations take relatively short amounts of time compared to the processing operations, determining the sequence of transportation operations in the system can be arbitrarily determined by the controller in real time. However, when the transportation operations take relatively longer times, they should be considered in the scheduling phase. Nevertheless, when an exact method is used to obtain the schedules, like the MIP models, including the transportation operations in the schedule building process will increase the complexity of the problem dramatically. Ramaswamy & Joshi [70] introduced the transporter as a new machine (resource) in the MIP model, with the transportation operations treated as processing operations. This approach guaranteed optimal overall schedules, but since each processing operation is usually associated with two transportation operations (to and from the processing machine), the resulting mathematical models became very complex and the solution time was prohibitive. They also proposed a second approach in which a transportation operations insertion heuristic was used after the schedule had been obtained using the MIP models. Although this heuristic produced fast and feasible results, a fixed efficient rule to break ties between jobs competing for the transporter was not defined.

Assuming that the transporter can handle only one job at a time, conflicts may arise at some points in time when the transporter is needed simultaneously by more than one job. In fact, some transportation operations must be carried out before others in order to achieve deadlock-free implementation of the schedule. For example, if a job is to be

delivered to a certain machine (or a buffer) that is already occupied by another job, then the latter job must first be moved to its next stage. As a result, some rules must be defined in order to resolve those conflicts while trying to achieve the best possible objective value of the schedule. In order to achieve that goal, a transportation operation insertion algorithm (TOI) is proposed. This algorithm can be utilized to insert the transportation operations in the schedules obtained using the MIP models and OI, or can be augmented in OI (Step 3.4 of the main algorithm) to consider the transportation operations in the schedule building process. Steps of TOI can be listed as follows:

Step 1. Determine the critical path (the sequence of operations forming the longest path in the schedule) of the current schedule.

Step 2. Determine the earliest scheduled time for a job to start or complete processing on a machine, to enter the system (if any), to leave the system (if any), to be moved to the buffer (if any), or to be moved from the buffer (if any).

Step 3. If more than one job shares the earliest scheduled time for a move, go to Step 4. Else, insert the transportation operation required to move the job associated with the earliest scheduled time. Go to Step 6.

Step 4. If one job is leaving a resource (machine or buffer) and another job is requesting that same resource, insert the transportation operation of the former first, and go to Step 6.

Step 5. If the jobs sharing the earliest scheduled time do not have a common resource in their moves, insert the transportation operation of the job (operation) *on the critical path.*

Step 6. Update the starting and ending times of all the processing operations in the current schedule and update the critical path.

Step 7. If all the transportation operations are inserted, STOP. Else, go to Step 2.

Since the proposed MIP models and insertion algorithm are generally associated with time-related objective functions (MS and MFT), a good tie-breaking rule between jobs competing for the transporter would then be the existence of their corresponding operations on the critical path of the schedule. Operations existing on the critical path do not have any slack, thus deferring such operations will always result in deferring other directly and indirectly dependant operations.

3.5 Numerical Example

In this section, a scheduling problem is solved using the proposed MIP models and OI. The model earlier proposed in [70] for the deadlock-free scheduling problem when infinite buffers are present (RJI), is also solved here to obtain the best possible solution for the problem in order to compare the capacity requirements in the IBA and the CBA models. TOI is also utilized to illustrate the difference between, inserting the transportation operations after obtaining the schedule with CBA model, and while building the schedule using OI. All the MIP formulations are modeled and solved using

the commercial software LINGO v6.0[1], and OI is coded and solved using MATLAB v7.1[2] using a PC Intel Duo Core/1.66G with 1G DRAM

The example problem is randomly generated and it comprises a manufacturing cell with three machines that are visited by six jobs. The processing routes and times of jobs on each machine are shown in Table C.1 in Appendix C. First, the problem is solved using the MIP models. The RJI, IBS, and IB1 models of this problem are solved first. Note that if the RJP_3 model was solved, the solution would have been the same as that of the IBS model. The IBA and CBA models are then each solved while incrementing the capacity of the buffer(s), until the solution obtained by RJI is obtained. In the IBA model, the three intermediate buffers are assumed to have the same capacity.

The solutions obtained for the RJI, IBS, and IB1 models are shown in Figures 3.11, 3.12, and 3.13, respectively. The values obtained for the objective function of minimizing the MFT by the RJI, IBS, and IB1 models are 140.17, 148.67, and 144 time units, respectively. These solutions were obtained in 9, 2.5, and 3.5 seconds, respectively. Figure 3.11 shows how the RJI model assumed that a machine becomes immediately available after completing a job, whether the next machine in that job's route was available or not. For example, machine M_2 started processing job J_6 as soon as it completed processing J_1 at time 27, although J_1 did not start its next operation on M_3 until time 82. If in the actual system there is no buffer space, this schedule will be unfeasible to implement.

[1] Copyright © 2007 LINDO Systems, Inc.
[2] Copyright 1984 – 2007, The MathWorks Inc.

Figure 3.11: Solution of the RJI model of example problem

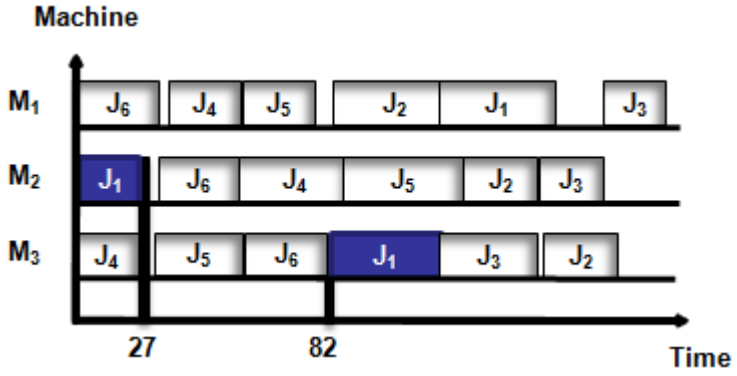


Figure 3.12: Solution of the IBS model of example problem

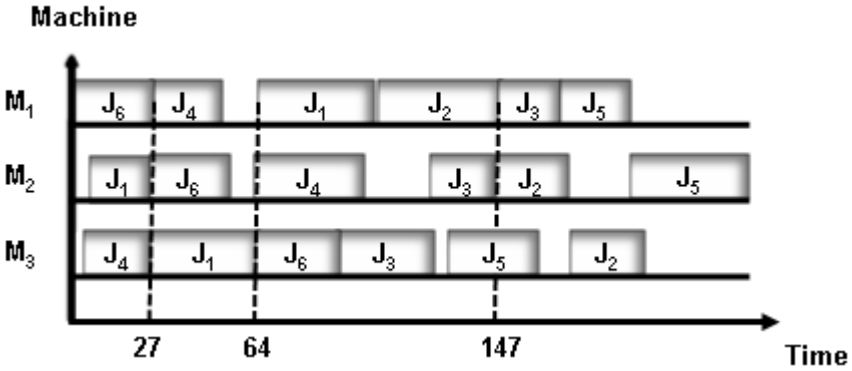
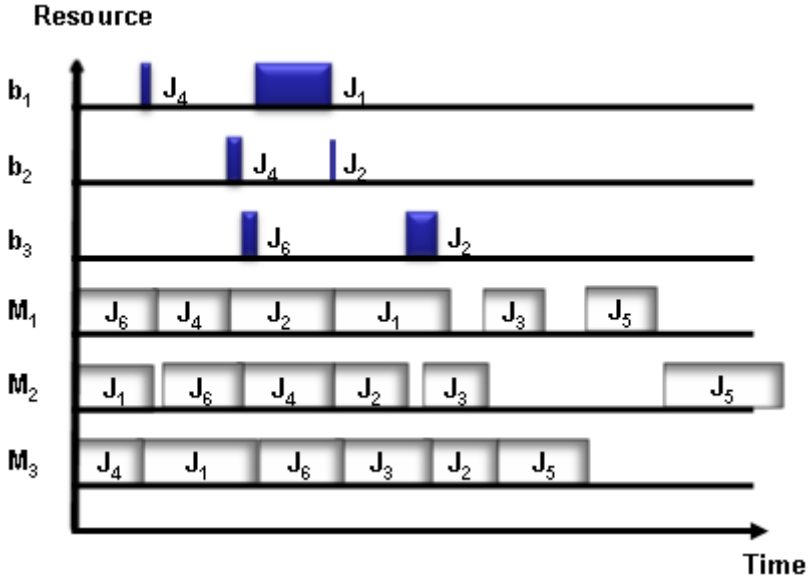


Figure 3.13: Solution of the IB1 model of example problem



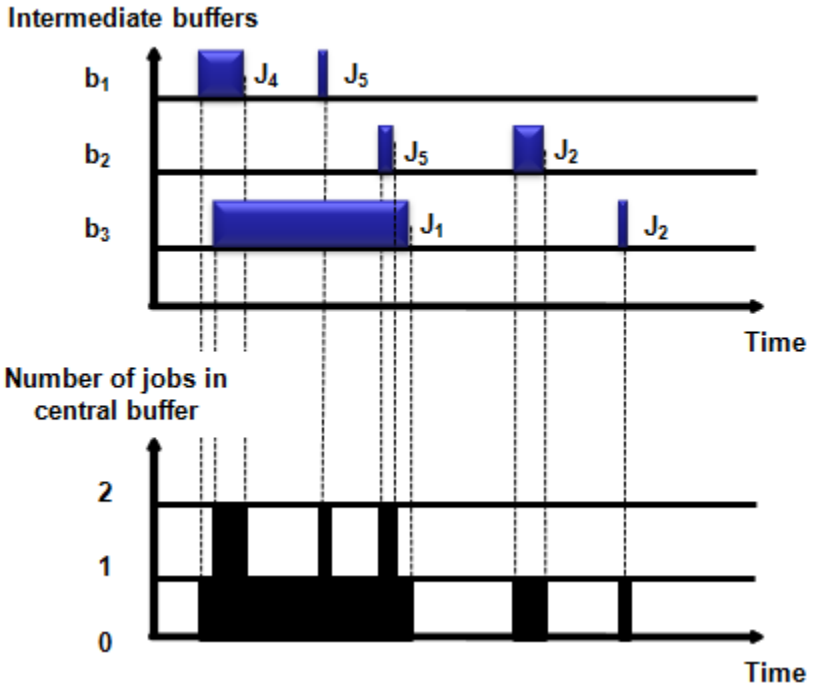
From the solution obtained by the IBS model (Figure 3.12), it can be seen that the intermediate buffers were used three times. First, at time 27, there was a circular wait between J_1, J_4, and J_6 on machines M_2, M_3, and M_1, respectively. To resolve such a circular wait, only one of these jobs needs to reside in the intermediate buffer preceding the next machine in its route (e.g. J_1 in b_3) momentarily until the other two jobs acquire their respective machines. In other words, the three jobs are swapped. The second time was again between J_1, J_4, and J_6 on machines M_3, M_2, and M_1 at time 64. The third time was to swap J_2 and J_3 between M_1 and M_2 at time 147.

In Figure 3.13, it can be seen how the IB1 model used the three intermediate buffers six times to hold different jobs, enabling other jobs to acquire the machines when needed. It can be noticed, however, that although this solution is superior to the one obtained by solving the IBS model, it is inferior to the one obtained by solving the RJI model. This is due to the restriction of having to start processing a job, residing in an intermediate buffer, on the machine as soon as it becomes available. For example, J_1 was released from b_1 as soon as J_2 was moved from M_1 to b_2.

The IBA and CBA models are used to obtain the solution obtained by the RJI model. The IBA model was able to obtain the same solution of 140.17 time units in 13 seconds using unit capacity intermediate buffers. On the other hand, the capacity of the central buffer in the CBA model had to be incremented to two in order to obtain this solution, which took a computational time of 58 seconds. The utilizations of buffers in the solutions obtained by the two models are shown in Figure 3.14. From this figure, it is clear that the IBA model only required unit capacity intermediate buffers because none of

these buffers were required simultaneously by more than one job at any time. The central buffer in CBA model, however, had to have a capacity of two in order to accommodate all the jobs in the buffer when required. This is because, as Figure 3.14 shows, J_1 resided in the buffer for a relatively longer time (from time 27 till time 82), which resulted in a dual utilization of the buffer every time J_4 or J_5 resided in it.

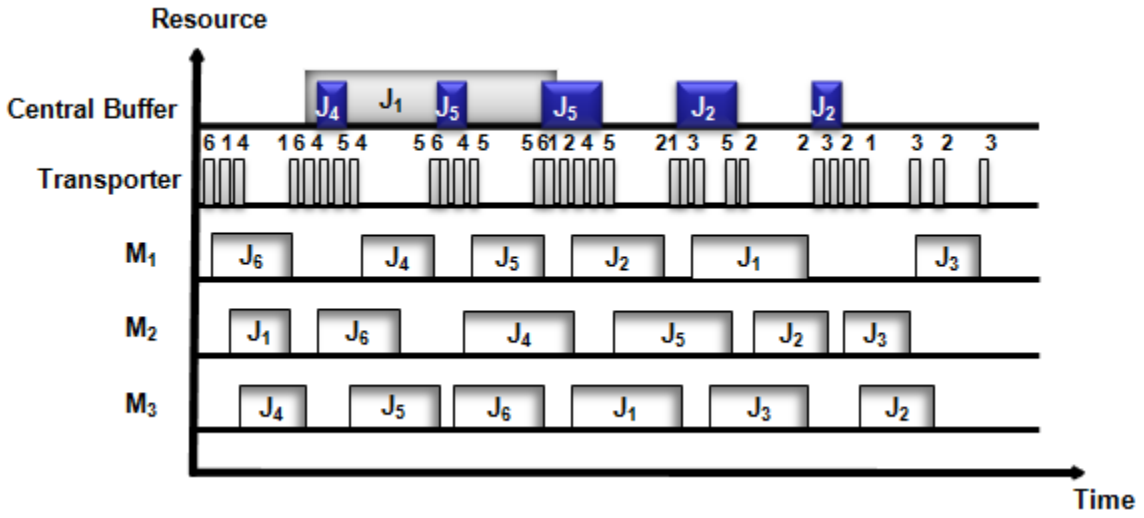
Figure 3.14: Illustration of buffer usage in the IBA and CBA models for the example problem



The above example also shows that the performance of the IBA model, when the capacity of the intermediate buffers is set to unity, can sometimes be better than that of the IB1 model that has inherent unit capacity intermediate buffers. From Figures 3.11 and 3.14, it can be seen that in the solution of the IBA model, J_1 was able to reside in b_3 while J_5 followed by J_6 were being processed by M_3. This however cannot be achieved by the IB1 model because of the aforementioned restriction in that model. Finally, TOI is

applied to the solution obtained by the CBA model. It is assumed that all the transportation operations take four units of time to acquire a job and deliver it to its destination. The obtained schedule has a MFT of 200 time units, and is shown in Figure 3.15.

Figure 3.15: Applying the transportation operations insertion algorithm on the CBA solution

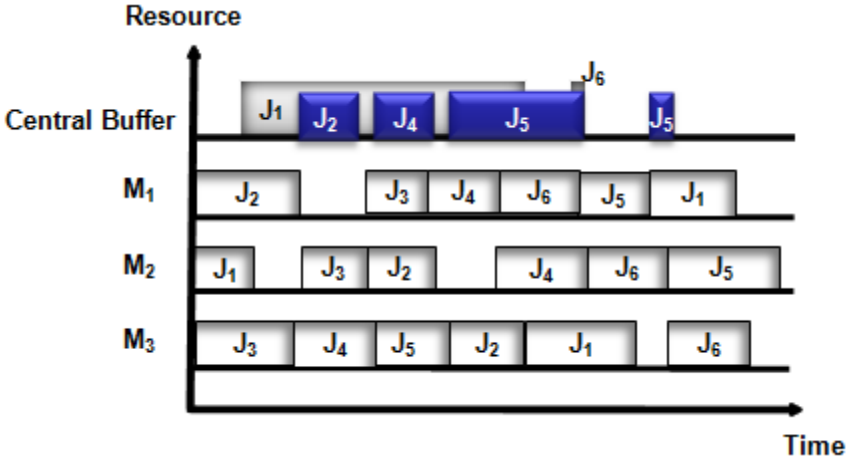


From the above figure, it can be noticed that the obtained schedule is still deadlock-free. In the original solution of the CBA model, J_4 resided in the buffer after being completed on M_3 until J_6 released M_1. This was to enable J_5 to start processing on M_3 as soon as possible. However, it could be noticed from Figure 3.15 that, after inserting the transportation operations, this step was no longer necessary. Since J_6 had already been transferred to M_2 before J_4 acquired the transporter to be transferred to the buffer, J_4 could have been directly transferred to M_1 instead. Nevertheless, the algorithm still had to apply this move since it was already scheduled in the original schedule. Although this indicates that this solution cannot be optimal, considering the transportation operations

from the beginning in the mathematical model to obtain the optimal solution would render it prohibitive to solve.

As for OI, it is used to solve the problem when there is a central buffer with a capacity equal to two twice; in the first time, without considering the transportation operations, and in the second, with TOI augmented in OI. The first schedule is obtained in 0.29 seconds and has a MFT of 148.83 time units (Figure 3.16). The order of insertion of jobs in the schedule using OI is: $J_1 - J_5 - J_4 - J_2 - J_6 - J_3$. From Figures 3.11 and 3.16, it can be noticed that OI failed to obtain the optimal solution having a MFT of 140.17 time units because the order of insertion of jobs considered J_2 before J_6. This resulted in scheduling J_2 as the first job on M_1 in the OI solution (the best schedule considering J_1, J_5, J_4, and J_2 only), which eventually resulted in the difference in the solution from the optimal.

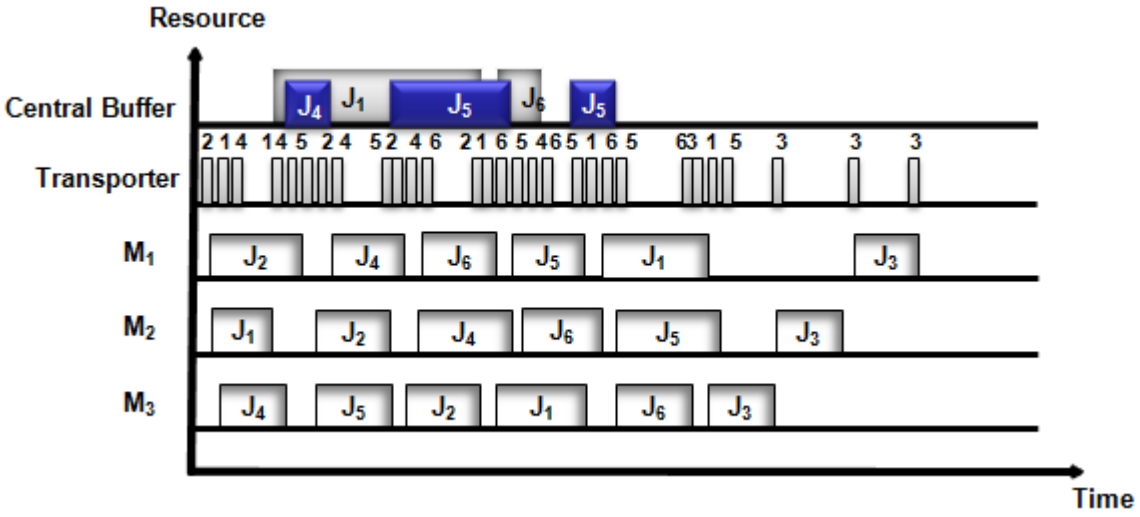
Figure 3.16: Using OI to solve example problem (buffer capacity = 2)



When TOI is augmented in OI, a schedule with a MFT of 190.67 time units (Figure 3.17) is obtained for the problem in 0.62 seconds. Compared to the schedule obtained by

inserting the transportation operations in the CBA model solution (Figure 3.15), it can be noticed that augmenting TOI in OI resulted in a better MFT in much lesser computation time. It can also be noticed that this schedule does not feature any unnecessary moves to the buffer like the previous solution, because the transportation operations are considered when such moves are scheduled.

Figure 3.17: Using OI augmented with TOI to solve example problem (buffer capacity = 2)



3.6 Performance Evaluation

In this section, the performances of the proposed MIP models and OI are evaluated. The MIP models are evaluated by performing a computational study, where a group of randomly generated problems are solved using the proposed models and the RJI and RJP_3 models. Performance is evaluated based on the solutions obtained and the computational time required to obtain these solutions. As for OI, its performance is evaluated by comparing it to those of previous approaches in solving a group of benchmark problems. Again, the evaluation is based on the quality of solutions and the computational time in some cases.

3.6.1 Computational Study for the MIP Models

Computational experiments have been a common method in literature to investigate the performances of mathematical models [61]. In the current study, ten groups of problems for ten different sizes of the problem on hand are studied. In each group, five problems are randomly generated and then solved using the two models proposed in [70], RJI and RJP_3, and using the four proposed models; IBS, IB1, IBA, and CBA. In the problems, the processing routes of jobs are randomly generated such that each job visits each machine once. Flexible routes are not considered in the current experiment because they were not modeled in [70]. Processing times of operations are randomly generated from a discrete uniform distribution over (20, 40) time units. The buffer capacities of the IBA and CBA models are incremented to obtain the best solution possible which is obtained by the RJI model.

The ten groups of problems are classified into two classes according to the number of machines in the system. The first class comprises of five-machine cells visited by 3, 4, 5, 6, and 7 different jobs. The second contains three-machine cells visited by 4, 5, 6, 7, and 8 different jobs. Although these problem sizes could be classified as small to fairly medium in the general job shop scheduling literature, based on experience and on previous literature they are realistic and represent real-world situations in FMSs that employ the cellular layout.

Table 3.2 shows the results obtained for the experiment. The first column shows the problem size indicated by *n* (number of jobs) x *m* (number of machines). The second column shows the model used. The third and fourth columns show the average objective

values obtained and average computational times required, respectively, for each problem size. For the RJP_3, IBS, and IB1 models, column five shows the average percentage deviation in objective value from the RJI model. Finally, for the IBA and CBA models, column six shows the average buffer capacities that were required to obtain the solutions. Note that a computational time limit of 3600 seconds [62] has been set for the current experiment. In other words, none of the models are allowed to exceed 3600 seconds in solving a problem (except for the RJI to acquire the best solutions).

From Table 3.2, the following observations can be made:

- The CBA model required more computational time than the set limit to solve problem sizes 7 x 5 and 8 x 3. Since it features the largest number of binary variables and constraints, the larger computational time was expected when the problem sizes got larger.
- On the other hand, although the RJI model features the minimum number of variables and constraints among all the models, it required more time than the set limit to solve problem sizes 7 x 3, 8 x 3, and 7 x 5. It can be noticed that the computational time of this model increased dramatically from problem sizes 6 x 5 to 7 x 5, and from 6 x 3 to 7 x 3. This may be due to the huge solution space that this model searches because of the minimized restrictions on the model.
- Although the IBS model obtained the same solutions as the RJP_3 model, it required the minimum computational time among all the models. In fact, the percentage reduction in computational time achieved by this model, when compared to the RJP_3 model, increased as the problem size increased until it reached nearly 57% for the 7 x 5 problem size.

Table 3.2: Results of computational study

Problem Size	Model	Objective value (time units)	Computational time (seconds)	Percentage deviation (%)	Buffer capacity
3 x 5	RJI	181.07	0		
	RJP_3	182.60	0	0.84	
	IBS	182.60	0	0.84	
	IB1	181.07	0	0	
	IBA	181.07	0		1
	CBA	181.07	0		1
4 x 5	RJI	190.8	0		
	RJP_3	194.95	0	2.1	
	IBS	194.95	0	2.1	
	IB1	191.3	0	0.26	
	IBA	190.8	0		1
	CBA	190.8	2.6		1
5 x 5	RJI	201.32	1.4		
	RJP_3	209.64	1.6	4.1	
	IBS	209.64	0.8	4.1	
	IB1	202.44	1.6	0.55	
	IBA	201.32	4.6		1
	CBA	201.32	23		1
6 x 5	RJI	208.77	13.4		
	RJP_3	229.00	20.8	9.69	
	IBS	229.00	16.4	9.69	
	IB1	211.37	13.6	1.24	
	IBA	208.77	68.4		1.4
	CBA	208.76	410.6		2.4
7 x 5	RJI	227.80	*		
	RJP_3	248.60	497.2	9.13	
	IBS	248.60	212.8	9.13	
	IB1	228.94	532	0.5	
	IBA	227.80	2326.2		1.2
	CBA		*		*
4 x 3	RJI	119.40	0		
	RJP_3	126.75	0	6.15	
	IBS	126.75	0	6.15	
	IB1	119.40	0	0	
	IBA	119.40	0		1
	CBA	119.40	0		1
5 x 3	RJI	134.36	0		
	RJP_3	140.04	0	4.22	
	IBS	140.04	0	4.22	
	IB1	135.28	0	0.68	
	IBA	134.36	0		1.6
	CBA	134.36	0.8		1.8
6 x 3	RJI	158.37	8		
	RJP_3	166.86	3.2	5.36	
	IBS	166.86	2.8	5.36	
	IB1	160.23	4	1.17	
	IBA	158.37	15		1.6
	CBA	158.37	59.4		1.8
7 x 3	RJI	168.49	*		
	RJP_3	178.66	24.6	6.03	
	IBS	178.66	25.4	6.03	
	IB1	169.57	30.6	0.64	
	IBA	168.49	523.4		2.0
	CBA	168.49	2977.4		2.2
8 x 3	RJI	173.88	*		
	RJP_3	187.60	362.5	7.89	
	IBS	187.60	292.5	7.89	
	IB1	178.50	457.75	2.65	
	IBA	173.88	1872.5		2.6
	CBA		*		*

*** Average computational time exceeded 3600 seconds**

- The IB1 model performed quiet satisfactorily in terms of both solution quality and computational time. On one hand it required computational times quiet comparable to those required by the RJP_3 model, and on the other it obtained solutions with deviations in objective values from the best possible solutions of RJI, IBA, and CBA, in the range of (0 to 2.65%) only. The RJP_3 and IBS models, however, reached up to 9.69% deviations from these best solutions.
- The IBA model was capable of obtaining the best solutions for all problem sizes within the allowed computational time limit.
- The CBA model generally required up to four or five times more computational time than the IBA model. However, it should be noted that for the CBA model, the buffer capacity requirement figures shown in Table 3.2, represent all the buffer space required in the cell. However for the IBA model, these figures represent the buffer capacity requirement for each intermediate buffer in the system.

3.6.2 Comparative Study for OI

A number of approximate approaches have been proposed in literature to solve the deadlock-free scheduling problem (Chapter 2). Among these approaches, different assumptions and problem parameters have been considered. Consequently, comparisons will be classified according to the solved benchmark problems, and in each case, the parameters considered in the other approach will be defined. The objectives of minimizing MS and minimizing MFT are both considered.

The first problem is a 4J x 3M problem and was introduced and solved in [70], [53], [54], and [59]. The problem was solved in these studies under different objectives and considering different system parameters. The next problem can be found in OR Library[3] under the name *la01*. This is a 10J x 5M problem and was solved in [57] considering the presence of a central buffer with a capacity equal to five. The last two problems are a 6J x 6M and a 10J x 10M problems that can be found in OR Library under the names *ft06* and *ft10*, respectively. These problems were solved in [51], [53], and [56] assuming the absence of buffer space and transporters, with the objective of minimizing MS. The best solutions obtained for these problems using the above approaches and using OI, along with the solution times of OI are shown in Table 3.3.

It should be noted that for problems *ft06* and *ft10*, the approach proposed in [51] had to be solved 20 times to obtain the best solutions shown in Table 3.3. For the 20 trials, this approach obtained an average makespan of 72 time units for problem *ft06*, and 1310 time units for problem *ft10*. It was also noted in [53] that the approach in [51] required over 200 CPU seconds on an AMD Duron 1GHZ processor to get a good solution. In addition, in [53], it was mentioned that the CPU times required to solve problems *ft06 and ft10* were almost negligible. However, the times required to construct the PN models of these problems and to locate the minimal siphons to prevent deadlocks were not mentioned. Furthermore, as in [51], the solution obtained in [57] for problem *la01,* was the best among 20 solutions (having an average makespan of 673 time units) and took a CPU time of 70 sec on a 500 MHz processor. Finally, the solution of *ft10* obtained in [56]

[3] OR Library, URL: http://msemga.ms.ic.ac.uk

was as well the best among other solutions that were obtained by varying a weight function. These solutions had objective values of 1319, 1333, and 1391 time units.

Table 3.3: Performance comparison results for OI

Problem	Approach proposed in	Nature of approach	Objective	Buffer	Transporter considered	Objective value	OI	
							Objective value	CPU time (sec)
'4Jx3M'	[70]	MIP	MFT	NO	NO	301.50	**301.50***	0.05
				YES	NO	287.75	**272.75***	0.13
				NO	YES	332.50	**332.50***	0.11
		MIP & heuristic		NO	YES	336.20	**332.50***	0.11
	[54]	PN & DP	MS	NO	YES	663	**560***	0.10
	[53]	PN & BS	MS	NO	NO	593	**512***	0.05
	[59]	Geometric approach & TS	MS	NO	NO	512	**512***	0.05
				YES	NO	502	**433***	0.14
				NO	YES	560	**560***	0.10
la01	[57]	TD & GA	MS	YES Cap. =5	NO	666	**666***	8.58
ft06* | *ft10	[51]	PN & GA	MS	NO	NO	69 | 1252	**69*** | 1324	0.70 | 22.85
	[53]	PN & BS	MS	NO	NO	69 | 1331		
	[56]	Automata & A*	MS	NO	NO	69 | 1319		

*** Optimal solution**

3.7 Conclusions

In this chapter new MIP models and an operations insertion algorithm (OI) were proposed to solve the deadlock-free scheduling problem with limited capacity buffers. In addition, a transportation operations insertion algorithm (TOI) was proposed to either insert transportation operations after obtaining the best schedules, or to augment these operations in the schedule building process when using OI.

The novelty in the MIP models is that they provided constraints that could represent and utilize the available buffer space, which was previously considered unachievable. These constraints further prevented any job from blocking a machine while waiting for the availability of the next machine in its route, and thus ensured the deadlock-freeness of the resulting schedules. The performances of the proposed models were compared to those of the models proposed in [70]. The computational results indicated that the IB1 model featured the best overall performance among all the models when considering objective function value and computational time together. The IBS model can obtain the same solution quality obtained by RJP_3 in a considerably less computational time. The buffer space capacity can be set arbitrarily in the IBA and CBA models. Although this comes at the expense of the required solution time to solve these models, it can be very beneficial in the system design stage, where solution time is not a major concern. These models can be solved to determine the minimum buffer capacity requirements that guarantee the optimal performance of the system. To optimize the performance of existing systems, trade-offs should be made between the solution quality and solution time when choosing between the proposed models.

As for OI, it utilized rank matrices in generating and evaluating schedules, and in detecting and preventing the occurrence of circular waits. It could handle a wide variety of parameters in the deadlock-free scheduling problem; jobs with alternative routes, jobs with different lot sizes, systems with no buffer space, and those with a limited central buffer space. In addition, it was shown how augmenting TOI in OI can provide schedules with better objective values than those obtained by inserting the transportation operations in optimal schedules. The performance of OI was compared to the performance of a number of approaches earlier proposed in literature in solving a group of benchmark problems. The results showed that in most of the cases, OI either obtained the same or better results than those achieved by other approaches in a timely and efficient manner. Due to time and memory space considerations, the proposed MIP models can be used to solve small and fairly medium-sized problems to optimality, whereas OI can be used to solve larger problems.

CHAPTER 4:

Reactive Scheduling

4.1 Introduction

As mentioned earlier in Chapter 2, a production system is a real world dynamic system subject to many disruptions. These can be external to the system, like arrival of new jobs (orders), cancellations of orders, and due date changes, or internal, like machine breakdowns and process time variations. Such disruptions should be accounted for, and reacted to efficiently, in real time to guarantee an efficient and stable performance of the system. Relying on the supervisory controller solely to react to these disruptions can most of the time realize a *safe* (deadlock-free) operation of the system. However, the *efficiency* in terms of production objectives, and the *stability* that reflects a less disturbed production flow, can only be addressed using a higher level reactive scheduler.

In a job shop environment, system resources like machine tools and fixtures are usually delivered to the corresponding machines based on the set production schedule. Furthermore, jobs may be set on pallets in a queue and ordered according to the defined sequence of operations in the schedule. Consequently, deviations in the set schedule may result in substantial costs. These include carrying costs for early delivered material, rush order costs for late delivery of tools and material, and costs incurred for re-sequencing

the ordered job queue, and reallocating the pallets [83]. In previous literature, a number of measures have been proposed for estimating system nervousness, or the amount of deviation from the original schedule [80, 110]. However, the most common measure of *deviation* from the original schedules has been the one used in [78], [81], [83], and [90]. It was measured as the normalized sum of absolute deviations of staring times of operations in the revised schedule from their starting times in the original schedule:

$$DEV = \frac{\sum_{i=1}^{n}\sum_{k=1}^{O_i} |\,(SR_{ik} - SO_{ik})\,|}{\sum_{i=1}^{n} O_i} \tag{4.1}$$

where, SR_{ik} and SO_{ik} are the starting times of operation k of job i in the revised schedule and the original schedule, respectively, and O_i is the number of operations of job i.

As stated earlier in Chapter 2, supervisory control (SC) approaches and deadlock-free reactive scheduling approaches that rely on either the automata or the PN formalisms, can handle system disruptions that do not add to the product mix of the system. Reacting to the arrival of new jobs using these approaches requires re-computing the whole control structure, which cannot be performed in real time. As for other disruptions, it has been shown earlier that the reactive scheduling literature lacks a generic approach that could handle a wide variety of disruptions in real time, while providing deadlock-free schedules.

This chapter will accordingly be divided into two main parts. The first (Section 4.2) will be dedicated to the new job arrival problem, where a detailed experimental study is

conducted to evaluate the relative performance of two approaches that may be pursued to react to this type of disruption; job insertion and total rescheduling (TR). In the second part (Sections 4.3 and 4.4), it will be shown how the proposed insertion algorithm (OI) can be extended to react to a number of disruptions through a generic deadlock-free reactive scheduling tool (GDRS). An experimental study will be also conducted to evaluate the performance of GDRS, via comparison with TR and the modified Affected Operations Rescheduling algorithm (mAOR) proposed in [90].

Throughout this chapter, along with the assumptions previously presented in Chapter 1 that defined the operational conditions of the considered flexible job shop systems, the following assumptions are adopted to define the reactive scheduling problem:

- An original deadlock-free production schedule that meets a certain objective criterion is already available.
- The problem is event-driven and is solved on a rolling horizon basis [111], or equivalently by taking snapshots of the systems when a disruption occurs [112]. This implicitly means that the actual time of occurrence of a disruption does not affect the problem, because the problem is solved each time a disruption occurs, considering only the remaining operations in the original schedule.

4.2 Arrival of New Jobs

Introducing a new job into a system, where a production schedule has already been established, requires the determination of time intervals wherein operations of this job

can be gainfully inserted. In the traditional reactive scheduling context, two approaches have been usually followed to react to the arrival of new jobs to the system [113]; job insertion (JI) and total rescheduling (TR). The JI problem was defined in [74] as finding a feasible simultaneous insertion of the operations of a new job (or jobs) into an existing feasible schedule of original jobs, while optimizing some objective criterion. TR on the other hand, implies solving the scheduling problem from scratch upon the arrival of a new job.

4.2.1 Job Insertion and Total Rescheduling

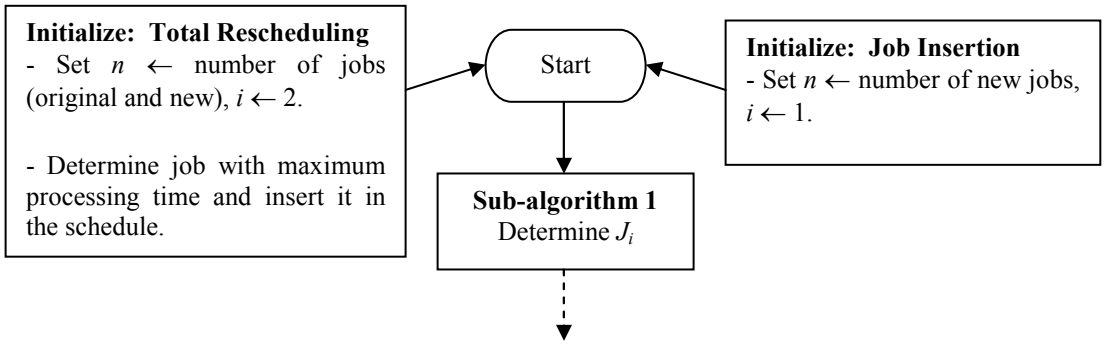
It has been noticed that most of the related literature have followed the JI approach rather than the TR approach in solving the new job introduction problem. In addition, some studies have concluded that, in reactive scheduling in general, TR can take significantly longer time and can result in more system nervousness while providing a better schedule (from the objective criterion's perspective). However, it has not been firmly stated when it is more beneficial to apply TR and which factors may affect these conclusions. In addition, depending on the solution methodology used and system properties being considered, these conclusions may not be valid all the time. Furthermore, any approach that has been proposed in literature to solve the deadlock-free *scheduling problem* can be utilized to solve the deadlock-free TR problem. However, the reported literature almost lacks an approach that can be used to solve the JI problem in a deadlock-free manner.

The insertion algorithm (OI) proposed in Chapter 3 is essentially a greedy algorithm that inserts into the schedule one job after the other in a deadlock-free manner. Hence, it can be utilized in the current context to apply both the JI and the TR approaches to add new jobs to the product mix. The only difference in application of OI for the two approaches is in the first step as follows (Figure 4.1):

Step 1. (Total rescheduling) Set $n \leftarrow$ number of jobs (original and new). Determine the job with the maximum processing time through the system and insert it in the schedule. Set $i \leftarrow 2$.

Step 1. (Job insertion) Set $n \leftarrow$ number of new jobs. Set $i \leftarrow 1$.

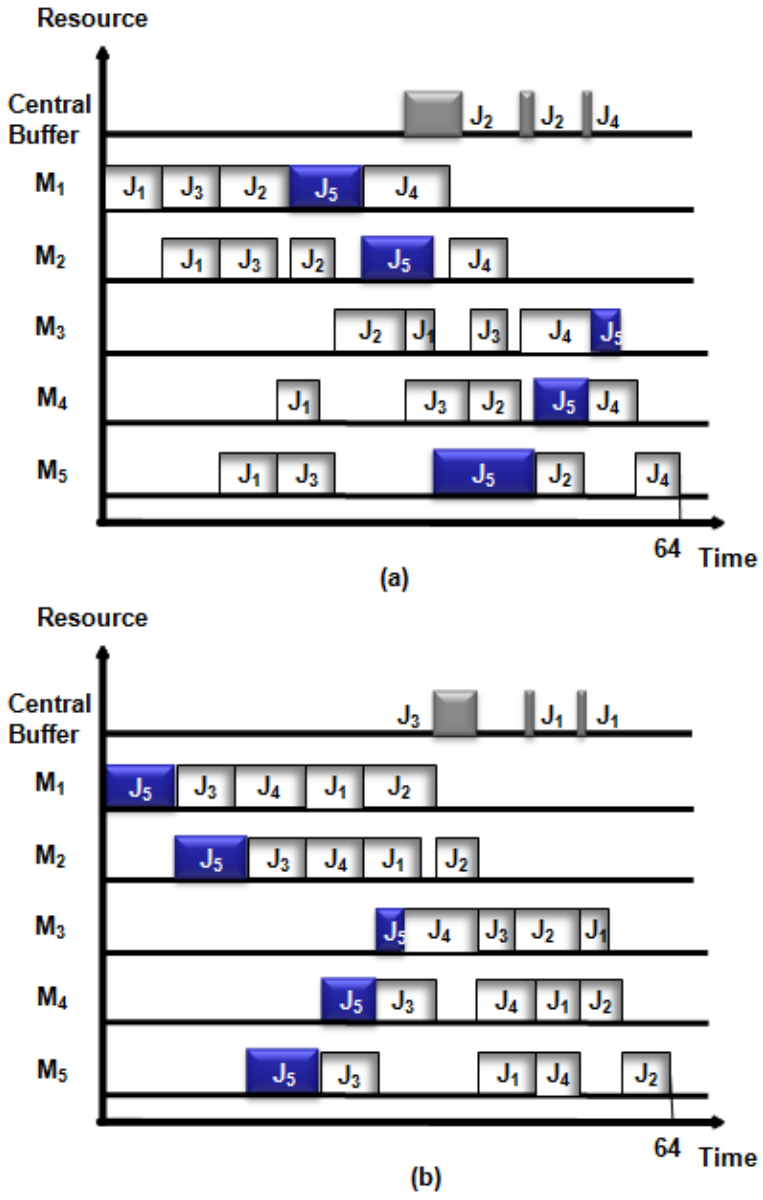
Figure 4.1: Application of OI to perform TR and JI



To illustrate the difference between the two approaches, the problem previously solved in [79] is solved here using JI and TR. In this problem, four jobs had been scheduled on five machines, when an additional job (J_5) was added to the system and had to be included in the schedule. The processing times and routes of the jobs on the machines are shown in Table C.2 in Appendix C. The schedule of the original jobs for this problem was obtained in that study with an optimal makespan (MS) of 60 time units, and that when J_5 was inserted was obtained with an optimal MS of 64 time units (Note that the

approach proposed in that study was a traditional approach, where infinite buffers were assumed). The reactive schedules obtained for this problem when applying JI and TR using OI are shown in Figures 4.2 (a) and (b), respectively. To account for the infinite buffers assumption in [79], a unit capacity central buffer is assumed to exist when solving the problem using OI.

Figure 4.2: Solution of comparison problem: a) JI solution; b) TR solution



From Figure 4.2, it can be noticed that both the JI and TR approaches obtained the optimal MS of 64 time units. Furthermore, in the TR schedule, the original relative sequences of the first four jobs on the machines are not preserved as in the JI schedule, which results in a considerable amount of deviation from the original schedule.

4.2.2 Experimental Analysis

In the following experiment, the relative performance of the JI and the TR approaches in solving a group of randomly generated reactive scheduling problems is tested and compared. The objective criterion considered in this experiment is minimizing the mean flow time (MFT) of the revised schedules. Furthermore, the effects of varying the values of some system parameters on the relative performance of the two approaches are studied. These parameters are *system size*, *number of new jobs*, *routing flexibility*, *presence of buffer space*, and *processing times of new jobs*. In addition to applying the JI and TR approaches, OI is also used to solve the randomly generated problems to obtain the initial schedules.

The system size factor (SIZ) is used to represent the number of machines and the number of original jobs in the system. Assuming that each job is processed once on each machine, this factor consequently represents the number of operations in the schedule, which if varied, may have a considerable effect on the revised schedules. The number of new jobs factor (NEWJ) affects the ratio of new jobs to original jobs, and thus increasing it may deteriorate the MFT of the JI approach. However, increasing it may also increase

the system nervousness resulting from the TR approach. As for the routing flexibility factor (RFLX), it is measured as the average number of alternative routes that a job can take through the system. The buffer space presence factor (BUFP) gives both approaches more flexibility in obtaining better schedules, since deadlock situations and blocking of machines can be resolved. Finally, changes in the levels of the operation times of the new jobs factor (OPTIM) may have considerable effects on the system nervousness and MFT. In fact, larger processing times of new jobs could affect system nervousness, especially in the TR approach, and can deteriorate the MFT obtained using JI. Table 4.1 summarizes the considered factors and the values of their levels.

Table 4.1: Considered system parameters and their levels

Factor	Code	Level	Value
System Size	SIZ	Low	Uniform*(3, 7) Jobs /uniform (3, 7) Machines
		High	Uniform (8, 12) Jobs /uniform (8, 12) Machines
Number of New Jobs	NEWJ	Low	20% of original jobs
		Medium	50% of original jobs
		High	100% of original jobs
Routing flexibility	RFLX	Low	1
		High	4
Presence of buffer space	BUFP	Low	No buffer
		High	Central Buffer present
Operation times of new jobs	OPTIM	Low	25% of original operations times
		High	100% of original operations times

*Numbers of jobs and machines at each level are generated from uniform distributions having the indicated ranges.

Having five factors, each with two levels except for the NEWJ factor with three levels, leads to 2 x 2 x 2 x 2 x 3 = 48 problem settings to study. In order to draw more accurate conclusions, and to statistically deduce the average effects of the factors on the performance measures, five replications are generated for each problem setting, resulting in a total of 48 x 5 = 240 test problems to study. Along with the considered factors, other parameters of the test problems are generated as follows:

- Processing times of operations of original jobs: generated from a uniform distribution between 70 and 110 time units, setting the average original operation time at 90 time units.
- Process plans of jobs: for each job, a random processing route is generated, ensuring that each job is processed once on each machine in the system.

4.2.2.1 Relative performance criteria

The performances of the two reactive scheduling approaches are evaluated in terms of three performance measures; system nervousness (deviation), MFT of the revised schedule, and solution time. Equation 4.1 is used to measure the deviation from original schedules. In the current experiment, it is assumed that every job is processed once on every machine. Consequently, the MFT in the system will increase as the number of machines increases with the increase in the system size factor (SIZ). Accordingly, this proportional relation has to be suppressed in order to obtain the obscured effect of changing the SIZ factor on the MFT measure. This is done, when applying the two reactive scheduling approaches, by dividing the resulting MFT by the total number of operations in the revised schedule to obtain the normalized mean flow time (*NMFT*).

Since this experiment is a comparative one, the values obtained for each performance measure by each of the two approaches are combined in one relative performance measure as follows:

- Relative Nervousness (RNERV) = $(DEV_{TR} - DEV_{JI})/$ average operation time
- Relative Mean Flow Time (RMFT) = $(NMFT_{JI} - NMFT_{TR})/$ average operation time
- Relative Solution Time (RSOLT) = (ST_{JI} / ST_{TR})

where the subscripts *TR* and *JI* refer to the criteria values obtained for total rescheduling and job insertion respectively, and *ST* indicates the solution time spent in solving the problem.

It should be noted that the value of RNERV (RMFT) is calculated as the difference between DEV_{TR} and DEV_{JI} ($NMFT_{JI}$ and $NMFT_{TR}$), because the nervousness (MFT) resulting from applying TR (JI) is usually higher than that resulting from applying JI (TR). It can also be noticed that the differences in deviation and normalized MFT obtained from both approaches are further divided by the average operation time to obtain the relative measures. This is done to eliminate the effect of the generated processing times of the operations on the performance measures. The objective is to obtain these measures in an absolute form, which can be utilized to evaluate any system setting.

4.2.2.2 Experimental results

In a factorial experiment there are two types of effects to study, the main effects and the interaction effects. The main effect of a factor corresponds to the average change in

the measured criterion produced by changing the level of this factor [114]. An interaction effect between two (or more) factors occurs when the change in the value of the criterion between the levels of a certain factor is not the same at all levels of the other factor(s); in other words, the effect of a certain factor depends on the levels of another factor(s). Main effects can generally be deduced from the results of a factorial experiment. However, significance of main effects and observation of interaction effects need more analysis to be determined. Analysis of Variance (ANOVA) provides such details.

Each of the 240 generated problems is first solved using OI to obtain an initial schedule for the original jobs with the objective of minimizing MFT. After including the generated new jobs with their associated operation times, the generated problems are then solved using JI and TR. The initial schedules are preserved in the JI solution, and are also utilized in both approaches to calculate the deviations in the revised schedules. The obtained averages for the three performance measures over the 240 problems are as follows:

- The average relative nervousness (RNERV) is 1.28.
- The average relative mean flow time (RMFT) is 0.10.
- The average relative solution time (RSOLT) is 0.80.

These results indicate that on an average, TR results in 128% more system nervousness per *original* operation per average operation processing time than JI. In addition, JI results in 10% more MFT per operation per average operation processing time than TR. Finally, using OI, JI takes 80% of the time required by TR to add new jobs in the production schedule. It should also be noted that the maximum solution time

encountered when solving the 240 problems was around three minutes. This limit was reached when applying TR to solve problems featuring high SIZ, NEWJ, and RFLX levels. That is, when solving the scheduling problem of 24 jobs on twelve machines, with each job having four alternative routes to choose from.

As mentioned earlier, having the performance measures in relative and absolute forms can serve as guidelines for selecting the appropriate reactive scheduling approach for any given setting. To illustrate, consider a system where 100 operations were originally scheduled, and that five more jobs, each requiring 10 operations, are to be added to this system, and that the average operation time in the system is 50 time units. Based on the above results, the following figures can be estimated to compare the application of the two approaches for this system setting:

- Difference in nervousness = 1.28 x 100 x 50 = 6400 more time units if TR is applied.
- Difference in MFT = 0.1 x 150 x 50 = 750 more time units if JI is applied.

The difference in nervousness, can be combined with the associated carrying or rush order, re-sequencing, and pallet reallocation costs (Section 4.1). Similarly, the difference in MFT can be combined with operational costs or the penalties for late product delivery. This would assist the decision maker in selecting the best approach based on actual cost figures. However, it should be noted that the system size and number of new jobs in the above example were set to represent an average sized system, based on the conducted experiments. Furthermore, other system parameters, like flexibility and buffer presence, were not taken into consideration. The following ANOVA results can account for these considerations.

4.2.2.3 ANOVA results

In general, ANOVA is conducted to determine if there are any significant main or interaction effects of the factors on the considered performance criteria (responses). This is done by comparing the variability among estimated effects (main and interaction) to the variability among replicate observations to calculate a certain ratio (*F*-ratio). A large *F*-ratio of an effect indicates the significance of that effect, and that it is larger than just to occur by chance or due to random errors. Together with the *F*-ratio, a *p*-value is also computed. The *p*-value represents the probability of making a Type 1 error and the null hypothesis assumes that the associated effect is not significant. The smaller the *p*-value, the smaller is the probability an error would be made by rejecting the null hypothesis when it is true. A cut-off value of 0.05 is often used, that is, reject the null hypothesis when the *p*-value is less than 0.05, indicating the significance of the effect [114].

Using the commercial statistical software MINITAB v13[4], ANOVA is conducted for the experiment on hand. The results indicate that changing the levels of SIZ, NEWJ, RFLX, and OPTIM, and the interactions SIZ x RFLX, SIZ x OPTIM, and NEWJ x OPTIM, all have significant effects on RNERV. In addition, the SIZ, RFLX, and BUFP factors along with the SIZ x BUFP and NEWJ x OPTIM interactions all have significant effects on RMFT. Finally, the NEWJ factor has a significant effect on RSOLT. To get a better understanding of these results, consider Figures 4.3, 4.4, and 4.5, which show the significant main effects. As for the significant interaction effects, Figures 4.6 and 4.7 display these effects on RNERV and RMFT, respectively.

[4] Copyright ©2006 Minitab Inc.

Figure 4.3: Main significant effects of factors on RNERV

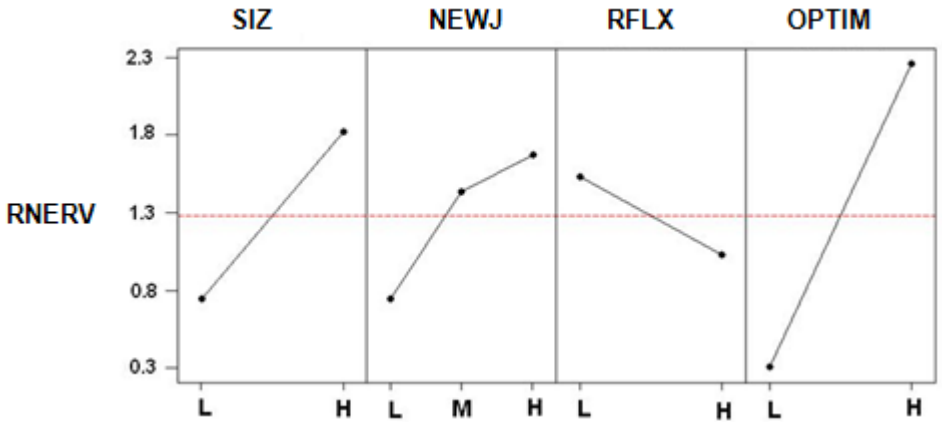


Figure 4.4: Main significant effects of factors on RMFT

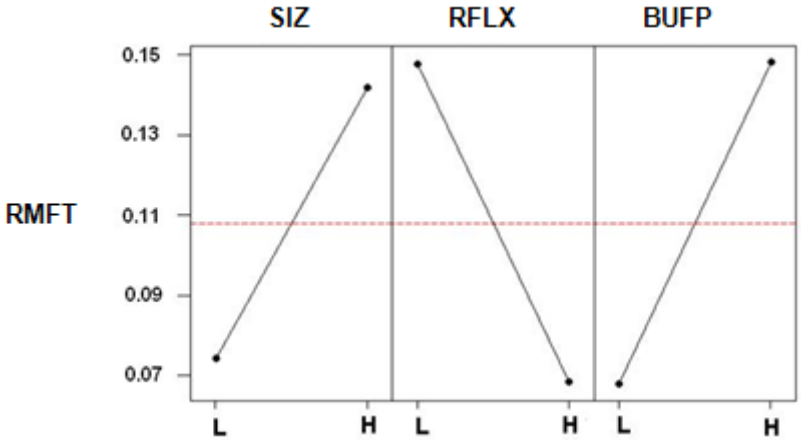


Figure 4.5: Main significant effects of factors on RSOLT

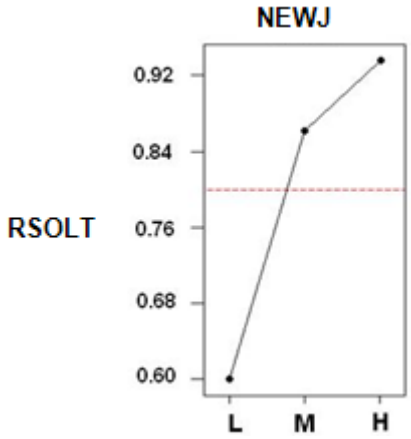


Figure 4.6: Significant interaction effects of factors on RNERV

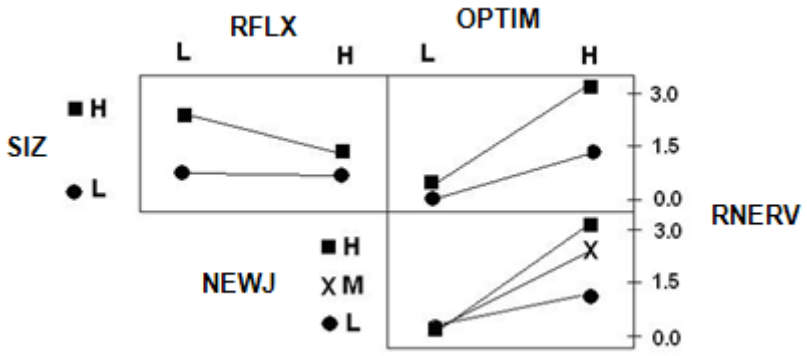
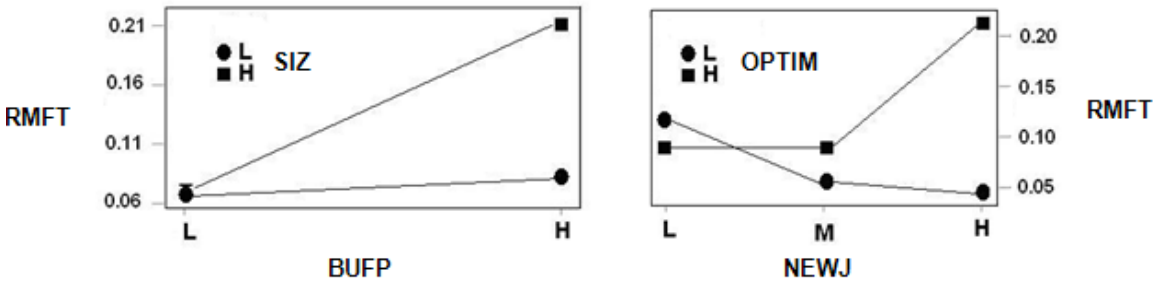


Figure 4.7: Significant interaction effects of factors on RMFT



The following observations can be deduced from the above figures:

- From Figures 4.3 and 4.6, it can be seen that RNERV significantly increases when OPTIM is high. In addition, this difference is further elevated when SIZ or NEWJ are at their high levels, to reach a threshold value for RNERV of 3.0. The significant increase in RNERV is a result of more operations having to be rescheduled. On the other hand, Figure 4.7 shows that when OPTIM and NEWJ are high, the difference in MFT between JI and TR becomes significant, and the value of RMFT reaches 0.22. However, when NEWJ is high and OPTIM is low, RMFT decreases to a value of 0.05, with no significant effect on RNERV.

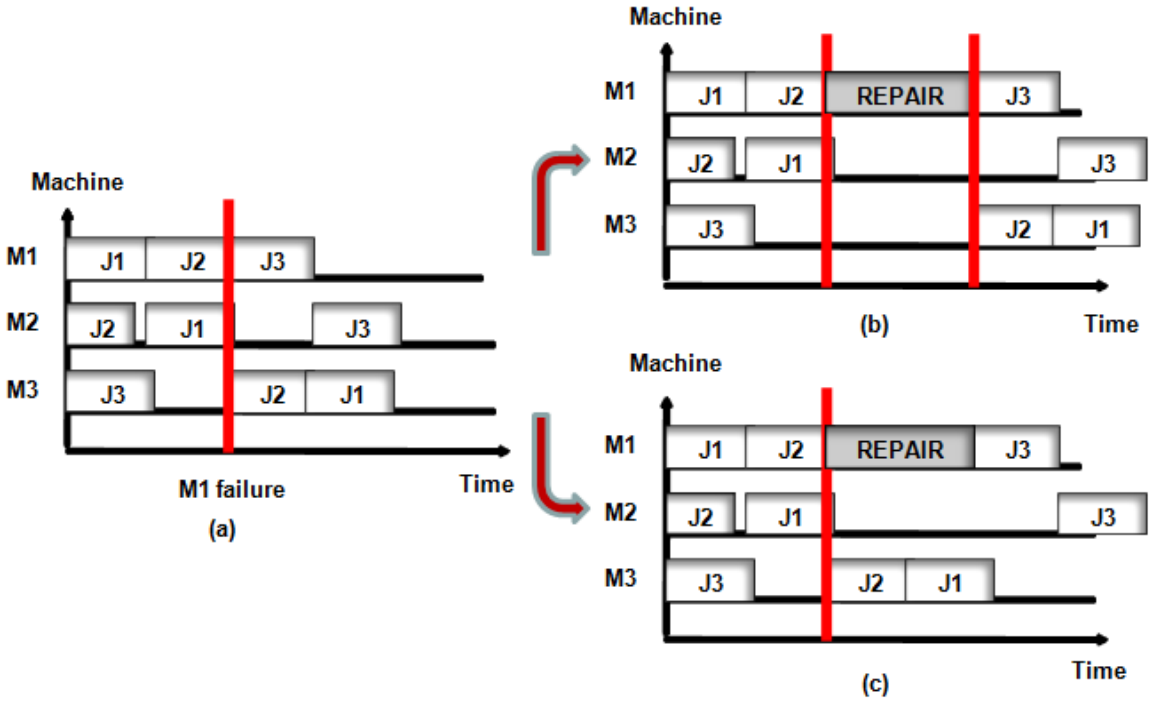
- Figures 4.4 and 4.7 show that when SIZ and BUFP are high, the difference in MFT between JI and TR becomes significant, such that the value of RMFT reaches 0.21. This is due to the inclusive nature of TR that makes better use of the buffer space when all the jobs in the system are considered for scheduling, especially in large systems. This eventually results in revised schedules with significantly better MFT than the ones resulting from JI.
- Figures 4.3 and 4.6 show that although RNERV increases significantly when SIZ is high, having routing flexibility (high RFLX) in the system suppresses this increase. In addition, Figure 4.4 shows that RMFT significantly decreases when RFLX is high. This indicates that having routing flexibility in the system on one hand reduces the system nervousness resulting from TR, and on the other it improves the MFT attained by JI.
- Finally, the effect shown in Figure 4.5 of NEWJ on RSOLT can be directly related to the ratio of the number of new jobs to the number of original jobs. In JI, increasing this ratio directly increases the solution time required, since this approach schedules only the new jobs. On the other hand, this ratio has a minor effect on the solution time of TR because in this approach, all the jobs (original and new) are nevertheless scheduled.

4.3 Generic Reactive Scheduling

As mentioned in Section 2.4.1, the Affected Operations Rescheduling algorithm (AOR) was proposed in [83], to react to machine breakdowns. In AOR, only the

operations directly and indirectly affected are pushed in time to account for the disruption, while keeping the original relative sequence of jobs on the machines unaltered. This algorithm was based on a binary branching algorithm, and it produced more efficient and stable reactive schedules when compared to right-shift-rescheduling (RSR) (Figure 4.8).

Figure 4.8: AOR and RSR; a) Machine 1 failure, b) reactive schedule using RSR, c) reactive schedule using AOR



In [90] and [78], AOR was used as a core for a modified AOR (mAOR) heuristic that defined generic schedule repair actions for a wide variety of disruptions. In the first study, mAOR was compared to RSR regarding the performance for four types of disruptions; machine breakdowns, arrival of new orders, process time variations, and urgency of existing jobs. The results showed that mAOR outperformed RSR in terms of both efficiency and stability.

In this section, the proposed operations insertion algorithm (OI) will be further extended to react to a number of system disruptions. Since the fundamental step of OI is the insertion of a single operation in a schedule in a deadlock-free manner, it can be adapted to react to wide variety of disruptions. It is employed through a generic deadlock-free reactive scheduling tool (GDRS) to react to machine breakdowns, process time variations, urgency of existing jobs, and cancellation of orders (jobs), as well as arrival of new jobs. In the sections to follow, the proposed set of steps required to react to each of these disruptions will be outlined along with an illustrative example. These will be followed by a comparative experimental study, in which the performances of GDRS, mAOR, and TR are evaluated and compared regarding five types of system disruptions.

4.3.1 Machine Breakdowns

A machine breakdown simply means that a machine will be unavailable for a period of time due to a sudden failure. In GDRS, the following steps are proposed to react to a machine breakdown:

1. Remove the job(s) whose operations are directly affected by the breakdown; i.e. operations on the broken down machine M_p whose starting times are less than the end of the downtime.

2. Insert an operation on M_p with duration equal to the downtime. This operation is inserted as the first operation on M_p. Note that according to the rolling horizon assumption (Section 4.1), the time of occurrence of the disruption in the original schedule is time zero in the revised schedule.

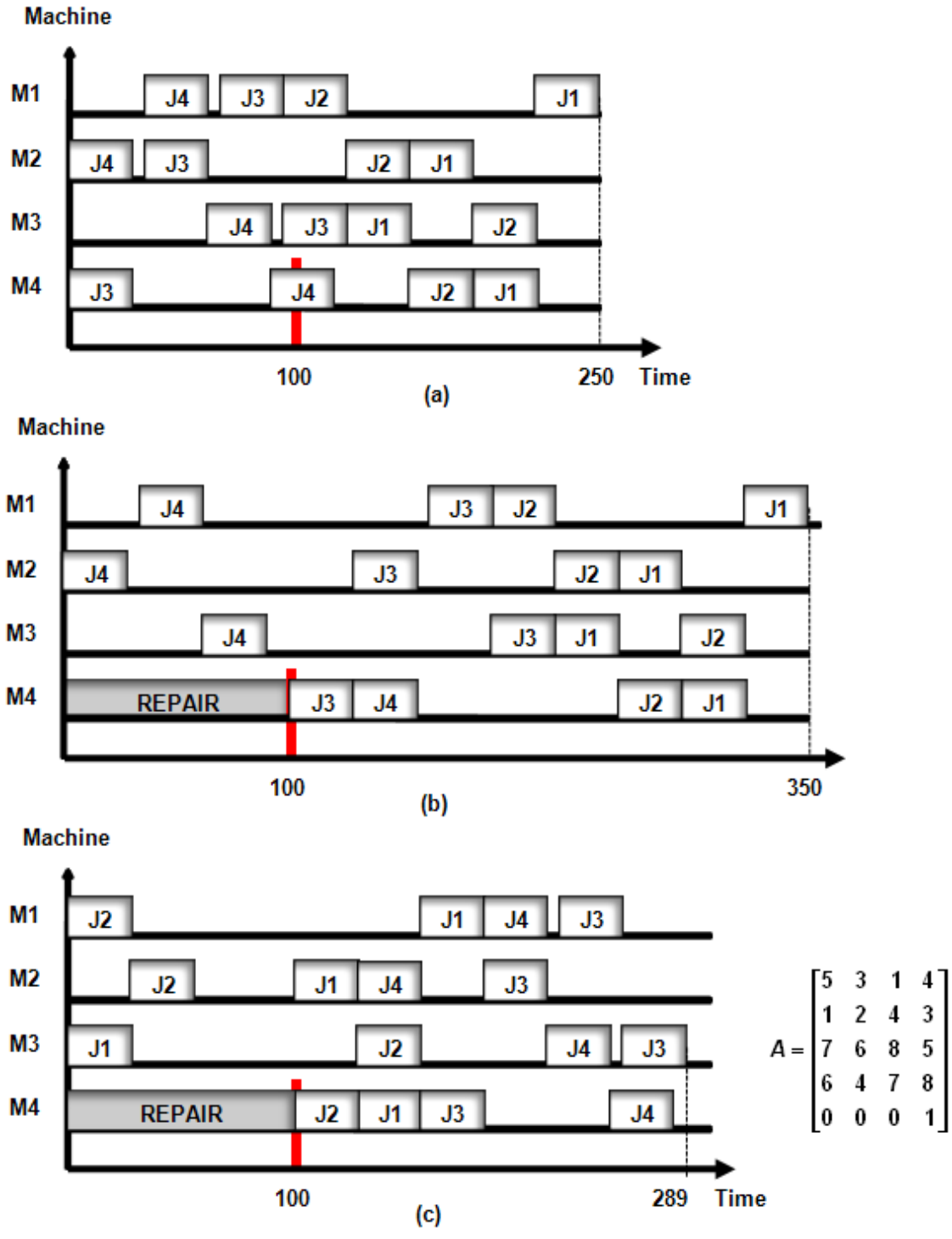
3. Re-insert the removed job(s) in the schedule, making use of alternative routes (if any) to avoid using M_p during its downtime, without altering the original relative sequences of operations of other jobs on all the machines.

To illustrate this, consider the following example. Figure 4.9(a) shows the Gantt chart of remaining operations in a deadlock-free schedule of four jobs on four machines with a MS (the performance criterion) of 250 time units, at the time of occurrence of a breakdown on Machine M_4. The downtime of the machine is 100 time units. Figure 4.9(b) shows the revised schedule with mAOR having a MS of 350 time units. According to GDRS, jobs J_3 and J_4 are directly affected by this downtime, and hence they are removed from the schedule, and re-inserted as shown in Figure 4.9(c), to achieve a makespan of 289 time units. Note that the machine downtime appears in the rank matrix as a fifth job with only one operation to be performed first on M_4.

4.3.2 Process Time Variation

This disruption is defined in [78] as the change in the end time of a process (operation). This change might be an increase or a decrease depending on the cause of the change. Since the defined sequences in a production schedule are merely based on the processing times of operations, the occurrence of such a change could, to a great extent, affect the logic behind which these sequences were obtained in the first place. Accordingly, in GDRS, the following steps are proposed to react to this type of disruption:

Figure 4.9: Machine breakdown: (a) Original schedule of remaining operations; (b) revised schedule using mAOR; (c) revised schedule using GDRS along with the associated rank matrix



1. Remove any job, whose operation has experienced a change in processing time, from the schedule.

2. Re-insert the removed job(s) in the schedule, with the new processing time(s), without altering the original relative sequences of operations of other jobs on all the machines.

Figure 4.10 (a) shows the Gantt chart of the remaining operations of a schedule, with a MS of 252 time units, at the time of discovery of a necessary increase in the processing time of J_3's operation on M_1 by 101 time units. Figure 4.10 (b) shows the revised schedule with mAOR, having a MS of 353 time units. Figure 4.10 (c) shows the revised schedule using GDRS with a MS of 325 time units after removing and re-inserting J_3 in the schedule.

4.3.3 Urgency of Existing Jobs

An urgent job is defined in [78] as an existing job that suddenly experiences an urgent demand or a due date revision that precedes its original completion time. In GDRS, the reaction to this disruption is carried out as follows:

1. Remove the urgent job from the schedule.

2. Re-insert the job, such that its completion time in the system precedes the revised due date, without altering the original relative sequences of operations of other jobs on all the machines.

Figure 4.10: Process time variation; (a) Original schedule of remaining operations, (b) Revised schedule using mAOR, (c) Revised schedule using GDRS

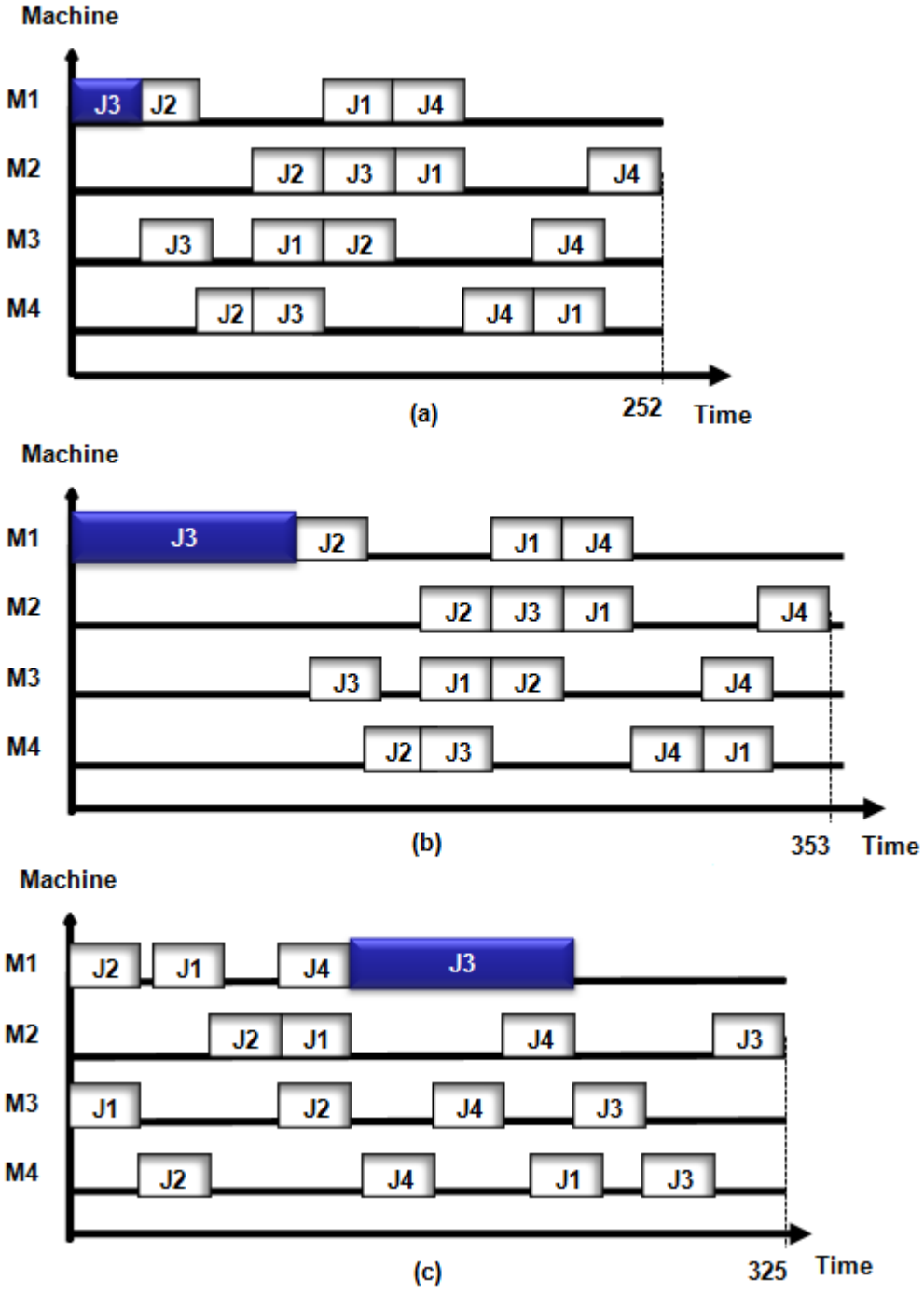


Figure 4.11(a) shows the remaining operations of a schedule, with a MS of 243 time units, at the time of receiving a revised due date for J_2 equal to its remaining processing time in the system; 128 time units. Figure 4.11(b) shows the revised schedule with mAOR, having a MS of 371 time units. Figure 4.11 (c) shows the revised schedule using GDRS after removing and re-inserting J_2 in the schedule with a MS of 304 time units.

Note that because the mAOR approach is a modification of the AOR approach that was originally developed to react to machine breakdowns, it reacts to the disruption of the urgency of an existing job, or to that of the arrival of a new job, by scheduling this job as the first job on all the machines that process this job in the revised schedule. This negatively affects both the efficiency and stability of the schedules revised using this approach due to the cumulative shifting action of other jobs on all the machines.

4.3.4 Order Cancellations

This disruption indicates that a job (order), already scheduled for processing in the shop, is not required anymore [78]. This disruption does not require the insertion of any operations; on the contrary, it requires the removal of some operations. In GDRS, the reaction to this disruption entails the following steps:

1. Remove the cancelled job from the current rank matrix.

2. Update the rank matrix, without altering the original relative sequences of operations of other jobs, and consequently the new starting and ending times of

operations. In other words, the schedule is *Left-shifted* as opposed to right shifting in RSR.

Figure 4.11: Urgent job; (a) Original schedule of remaining operations, (b) Revised schedule using mAOR, (c) Revised schedule using GDRS

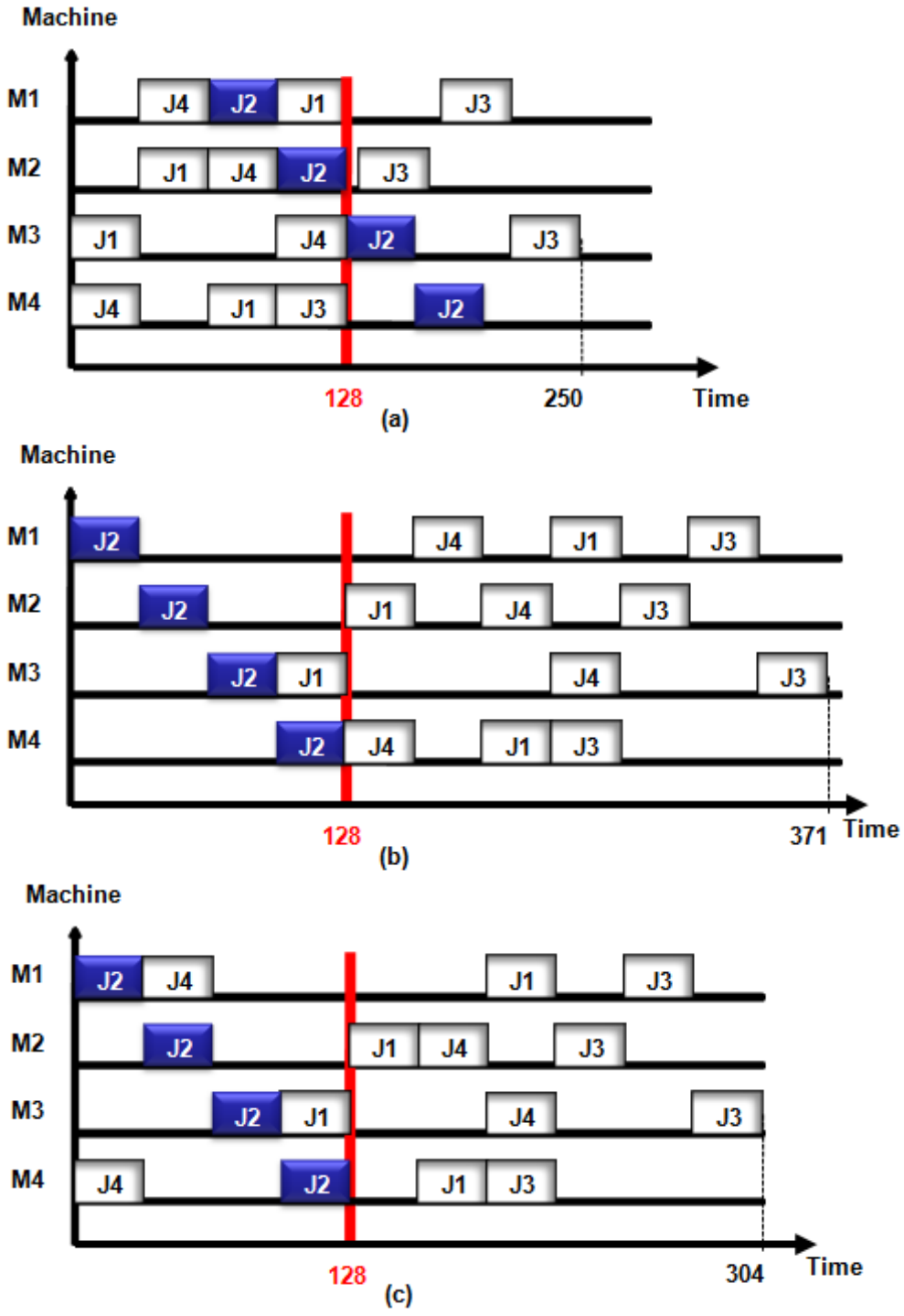
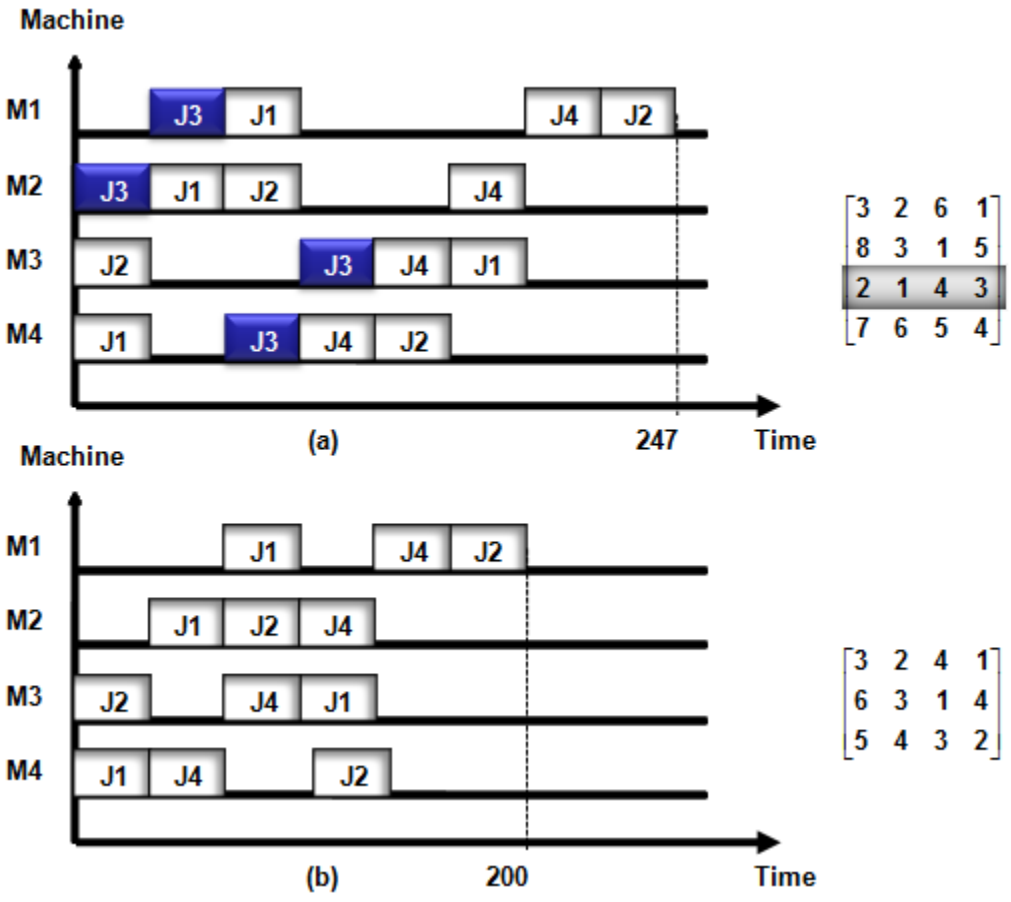


Figure 4.12(a) shows the remaining operations of a schedule, with a MS of 247 time units, at the time of receiving a cancellation order for J_3. Figure 4.12(b) shows the revised schedule using GDRS with a MS of 200 time units after removing J_3 from the schedule. Note that in mAOR, the reaction to this disruption only removes the cancelled job without any further repair actions [78, 90]. In other words, if mAOR was used in the shown example, the MS would have stayed at 247 time units.

Figure 4.12: Order cancellation; (a) Original schedule and rank matrix of remaining operations, (b) Revised schedule and rank matrix using GDRS



4.4 Comparative Analysis

In this section, an experimental study is conducted to compare the performances of the three discussed reactive scheduling approaches; the proposed GDRS, mAOR, and TR. Performance is evaluated in terms of efficiency and stability of revised schedules when reacting to five types of system disruptions; machine breakdowns, arrival of new jobs, process time variations, urgency of existing jobs, and order cancellations. Since there are five types of disruptions, five separate factorial experiments will be conducted for each type. Another objective of the experiment is to study the effects of changing a number of system parameters (factors) on the performances of the approaches. The design entails factorial experiments in which all the possible combinations of the levels of various factors are studied in each trial [114]. The problems studied along with the disruptions are randomly generated (Section 4.4.1). As in Section 4.2, in order to obtain the original best deadlock-free schedules of the remaining operations for the problems and to apply TR, OI is utilized. It should be noted that, although in Section 4.2 a dedicated experimental study was conducted to study the arrival of new jobs, this disruption is again considered in the current experiment mainly to compare the performances of GDRS to mAOR.

4.4.1 Experimental Design

After a careful search within the literature for an experiment similar to the current one, the experiment conducted in [83] appeared to be the most relevant. In that experiment,

performance comparison was conducted between three rescheduling approaches, RSR, AOR, and TR, where the effects of five experimental factors were studied; the rescheduling method, time of occurrence of the disruption, magnitude of the disruption, optimality of the original schedule, and the size of the original scheduling problem. The 'optimality of the schedule' factor is overlooked in the current experiment due to the hardness of obtaining optimal solutions while solving the deadlock-free scheduling problem for large problems. Accordingly, the factors considered in the current experiment are:

- Reactive scheduling method **(MTD):** This factor is selected to study the effect of changing the reactive scheduling method on the performance of the revised schedules.
- Size of the reactive scheduling problem **(SIZ):** This factor is a combination of the 'time of occurrence of disruption' and the 'size of the *original* scheduling problem' factors. It refers to the number of remaining operations in the schedule upon the occurrence of the disruption.
- Magnitude (size) of the disruption **(MAG):** For each of the five experiments, this factor is defined according to the associated type of disruption.
- Flexibility of system **(FLX):** This factor represents both the availability of routing flexibility and buffer space in the system. Routing flexibility is again defined by the available number of alternative routes for each job.

The MTD factor will feature three levels representing the three reactive scheduling methods considered. As for the rest of the factors, each factor will feature two levels as shown in Table 4.2. Note that in this table and in what follows, the terms MS and

schedule refer to the schedule of the remaining operations at the time of occurrence of the disruption. As for the disruptions, they are randomly generated in the problems as follows:

Table 4.2: Levels of experimental factors

Experimental Factor		Level 1 (Low)	Level 2 (High)
Problem size (SIZ)		Uniform† (3 , 7) Jobs Uniform (3, 7) Machines	Uniform (15, 20) Jobs Uniform (10, 15) Machines
Magnitude of disruption (MAG)	Machine breakdown	Downtime = 10% of MS	Downtime = 40% of MS
	Arrival of new job	Processing time/operation = 0.75 * average operation time	Processing time/operation = 1.25 * average operation time
	Process time variation	Increase by 10% of MS	Increase by 40% of MS
	Urgent existing job	Revised due date = 150% of processing time of job	Revised due date = processing time of job
	Order cancellation	Late scheduled job (last third of schedule)	Early scheduled job (first third of schedule)
Flexibility of system (FLX)		No alternative routes and no buffer capacity	Routing flexibility = 2/job Buffer size = 1 (small problem) or 2 (large problem)

† Numbers of jobs and machines are generated from uniform distributions having the indicated ranges

- Machine breakdown: a machine is randomly selected to represent the broken down machine, provided that this machine started processing on a job at time zero of the schedule. This is done to ensure the direct effect of the disruption on at least one job.
- Arrival of new job: a new job is randomly generated.
- Process time variation: an operation is randomly selected, from those that start processing at time zero on any machine, to vary its processing time.
- Urgency of existing job: a job is randomly selected from the schedule to revise its due date.

▪ Order cancellation: a job (order) is randomly selected to be cancelled according to the levels shown in Table 4.2.

Considering three factors in the factorial combinations, each with two levels, leads to $2^3 = 8$ problem settings to study for each type of disruptions. In order to draw more accurate conclusions, and to statistically deduce the average effects of the factors on the performance measures, five replications [83] are generated for each problem setting. This results in 40 test problems to study for each type of disruptions; 200 test problems in total. In the test problems, processing times of original operations are generated from a uniform distribution between 20 and 40 time units, setting the average original operation time at 30 time units.

Two performance measures are collected in this experiment; efficiency and stability of the revised schedules. In previous literature, efficiency was evaluated in terms of the value of the MS of the revised schedule relative to that of the original schedule. In this study, only the schedules of the remaining operations are considered. Consequently, the evaluation of efficiency is based on the makespans of the revised and original schedules of the *remaining* operations as follows:

$$EFF = (1 - \frac{MS_{rev} - MS_{org}}{MS_{org}}) \times 100\% \quad (4.2)$$

where MS_{rev} and MS_{org} are the makespans of the revised and original schedules of remaining operations, respectively.

Note that when reacting to the order cancellation disruption, this efficiency measure should result in values either equal to or higher than 100%, since the revised schedules are expected to obtain equal or lower makespans than the original schedules. As for the stability measure, it is evaluated in terms of the deviation of the revised schedule from the original schedule (Equation 4.1). Note that lower deviation values indicate higher stability and thus better performance of the reactive scheduling approach.

4.4.2 Experimental Results and ANOVA

The algorithms for the three reactive scheduling approaches are coded using MATLAB v7.1. Table 4.3 shows a summary of the average values of the performance measures obtained from solving the test problems using the three approaches for each type of disruptions:

Table 4.3: Average values of performance measures

Approach	Disruption									
	Machine breakdown		New job arrival		Process time variation		Urgent job		Order cancellation	
	EFF (%)	DEV	EFF (%)	DEV	EFF (%)	DEV	EFF (%)	DEV	EFF (%)	DEV
GDRS	87.7	99.9	85.2	17.9	79.8	78.9	92.4	46.6	107.2	15.47
mAOR	75.6	108.5	59.0	169.4	78.5	89.0	66.6	132.1	100.2	0.0
TR	91.1	159.0	88.8	95.8	88.7	122.3	96.1	57.7	113.6	62.4

From Table 4.3, it can be noticed that the proposed GDRS approach clearly outperforms mAOR in terms of both efficiency and stability for most of the considered disruptions. However, for the order cancellation disruption, mAOR results in less (zero) deviation from the original schedules, but with a negligible improvement in schedule efficiency. In addition, the difference in performance between the two approaches is slight when reacting to the process time variation disruption. As for GDRS and TR, it can be noticed that, with the exception of the process time variation disruption, GDRS tends to result in revised schedules with efficiencies close to those obtained by TR. On the other hand, GDRS clearly outperforms TR in terms of the stability of the revised schedules for all the considered disruptions. As for the ANOVA, focus is directed towards testing the significance of the following effects:

- Main effect of changing the levels of MTD on the two performance measures (efficiency and stability). This is carried out by conducting a one-way ANOVA test between the MTD levels on the EFF and DEV measures.
- Main effects of changing the levels of the other experimental factors (SIZ, MAG, and FLX) on the two responses.
- Interaction effects between MTD on one hand, and SIZ, MAG, or FLX on the other.

Using the commercial statistical software MINITAB v13, the results for the current ANOVA experiments are obtained. Table 4.4 shows the *p*-values for the main and interaction effects of the considered factors on the two performance measures for the five types of disruptions (significant *p*-values are indicated using bold face).

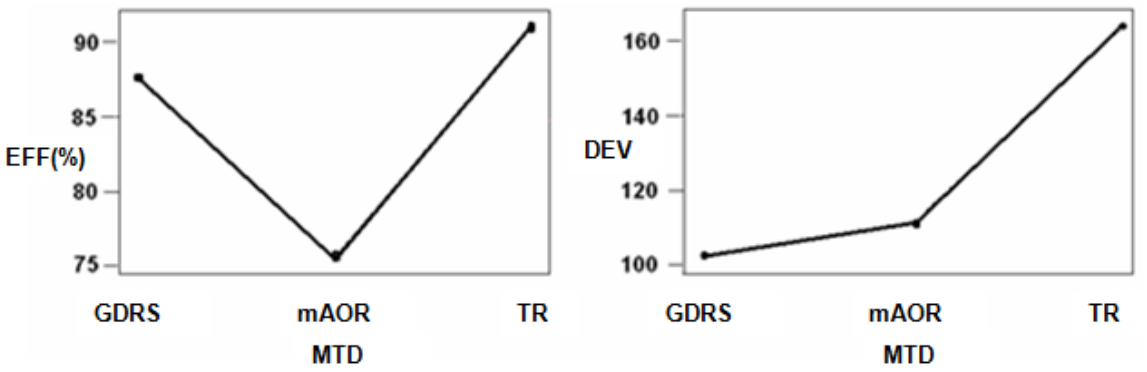
Table 4.4: *p-values* for the ANOVA experiments

Effect	Machine Breakdown		New job arrival		Process time variation		Urgent existing job		Order cancellation	
	EFF	DEV	EFF	DEV	EFF	DEV	EFF	DEV	EFF	DEV
MTD	**0.000**	**0.022**	**0.000**	**0.000**	**0.004**	**0.048**	**0.000**	**0.000**	**0.000**	**0.000**
SIZ	0.337	0.000	**0.000**	0.000	0.114	**0.000**	0.051	0.000	0.000	0.001
MAG	0.000	0.000	0.000	0.000	0.000	0.000	**0.020**	0.102	0.306	0.339
FLX	**0.011**	**0.000**	**0.000**	**0.027**	0.360	**0.000**	0.180	0.496	0.207	0.750
MTD*SIZ	**0.004**	**0.000**	0.171	**0.000**	0.142	0.128	**0.029**	**0.010**	**0.010**	**0.000**
MTD*MAG	**0.000**	**0.003**	**0.009**	**0.000**	**0.000**	**0.003**	0.098	0.072	0.615	0.743
MTD*FLX	0.229	0.102	0.342	0.358	0.686	0.119	0.297	0.106	0.573	0.899

4.4.2.1 Machine breakdowns

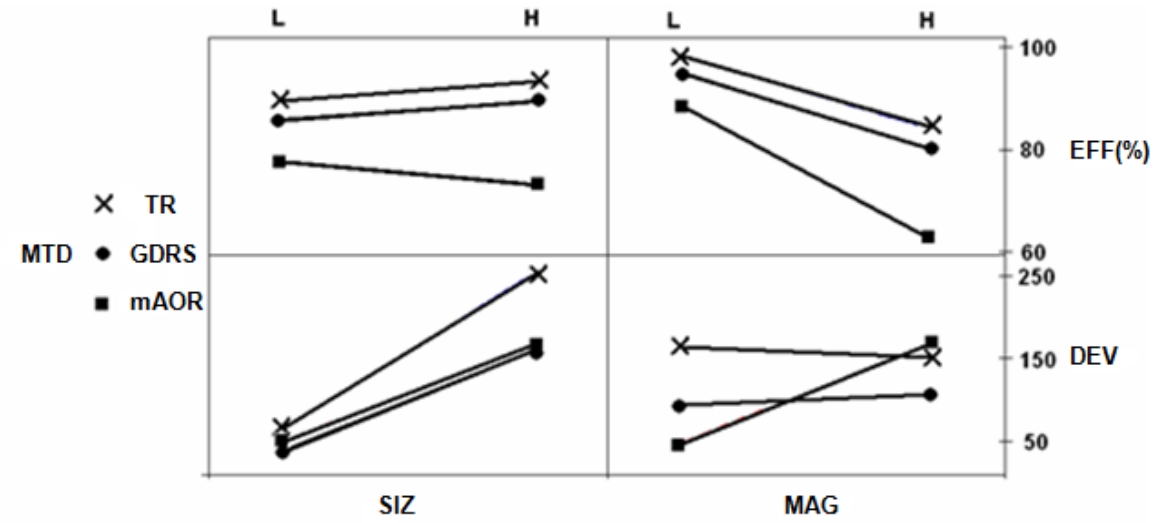
For the machine breakdown disruption, the results obtained for the MTD factor from the one-way ANOVA test are shown in Figure 4.13. These results show that the efficiencies of the schedules revised using GDRS and TR were significantly higher than those revised by mAOR. This is because in GDRS, the starting times of the indirectly affected jobs are not shifted unnecessarily like mAOR. The results also show that there is no significant difference in efficiency between TR and GDRS. This is because, unlike mAOR, GDRS does not right-shift the operations to account for the repair time of the broken-down machine. It only reschedules the directly affected jobs, while making use of alternative routes (if any), without deferring any operations on the other machines. This minimizes the changes in the makespan of the schedule. As for stability, the results show that, although GDRS results in less deviation than mAOR, this difference is not significant. Furthermore, GDRS and mAOR obtain significantly lower deviations than those obtained by TR.

Figure 4.13: One-way ANOVA results for the machine breakdown experiment



As for the other experimental factors, results from Table 4.4 indicate that, introducing flexibility in the system increases the efficiency and stability of the revised schedules. This is because of the alternative routes and the availability of buffer space that can be utilized to reduce blocking of machines. As for the interaction effects, it can be seen in Figure 4.14 that unlike the TR and GDRS approaches, mAOR tends to suffer from a decrease in efficiency as SIZ increases. Results also show that, mAOR is more negatively affected than TR and GDRS by the increase in MAG. As for the stability of the revised schedules, Figure 4.14 shows that TR is more negatively affected by the increase in SIZ than GDRS and mAOR. Furthermore, although GDRS and TR tend to maintain stability with the increase in MAG, mAOR is negatively affected to the extent that it shows more deviation than TR when large machine downtimes are experienced. This may be due to the shifting approach of mAOR, which eventually accumulates more deviation when the downtime is increased.

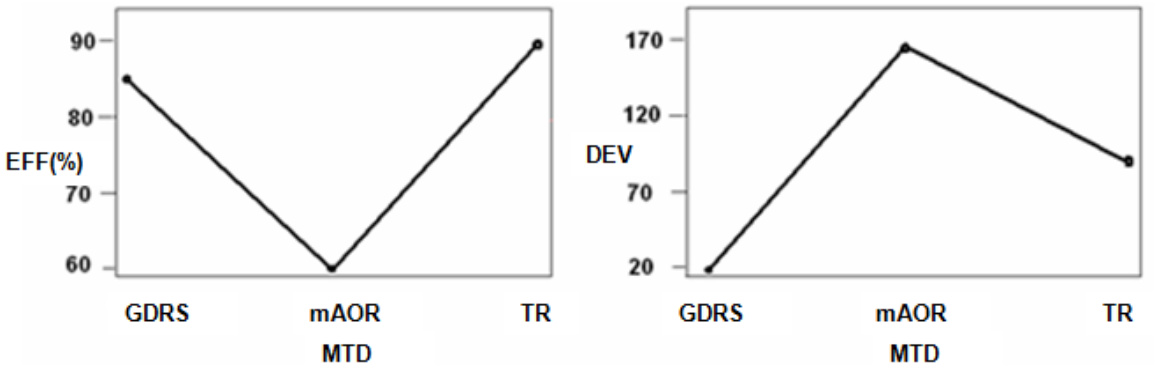
Figure 4.14: Significant interaction effects for the machine breakdown experiment



4.2.2.2 New job arrivals

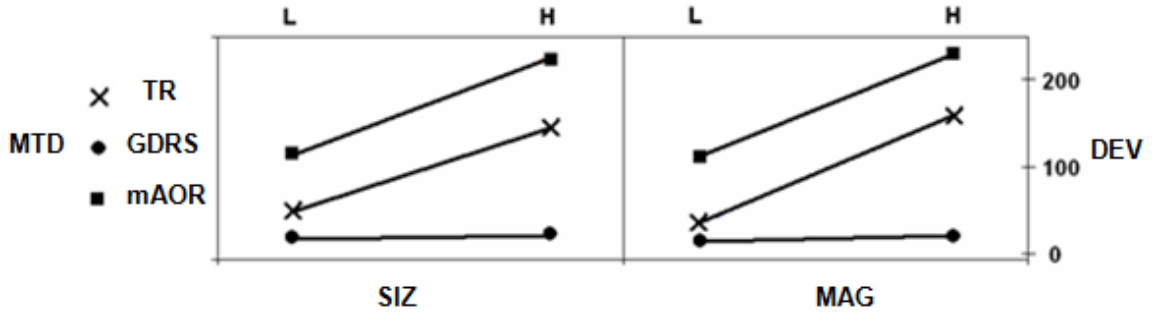
Figure 4.15 shows the one-way ANOVA results for the new job arrival disruption. It can be seen that the efficiencies of the schedules revised using GDRS and TR were significantly higher than those revised by mAOR, and that there is no significant difference in efficiency between TR and GDRS. This is because GDRS inserts the operations of the new job on the machines at the positions that result in the least effect possible on the makespan of the schedule. In addition, GDRS results in significantly better stability than TR that in turn features significantly better stability than mAOR. This deterioration in stability of schedules obtained by mAOR is due to the accumulative effect of shifting the starting times of original jobs on every machine that performs an operation on the new job.

Figure 4.15: One-way ANOVA results for the new job arrival experiment



From table 4.4, it can be noticed that increasing SIZ improves the efficiency of the schedules, which is probably due to the decreased ratio of new jobs to original jobs; in both levels of SIZ, the number of new jobs is fixed at one. Furthermore, increasing FLX again improves both the efficiency and stability of the revised schedules. Analysis of interaction effects shows that the efficiency of mAOR is more negatively affected by the increase in MAG than GDRS and TR. It also indicates that while the stability of revised schedules obtained by mAOR and TR is negatively affected by the increase in SIZ and MAG, GDRS tends to maintain the same level of stability of the revised schedules as shown in Figure 4.16.

Figure 4.16: Significant interaction effects for the new job arrival experiment



4.2.2.3 Process time variations

Figure 4.17 shows the one-way ANOVA results for the process time variation disruption. These results show that TR obtains revised schedules with significantly better efficiencies than GDRS and mAOR. On the other hand, the deviation resulting from GDRS is significantly lower than that resulting from TR, but not significantly lower than mAOR.

Figure 4.17: One-way ANOVA results for the process time variation experiment

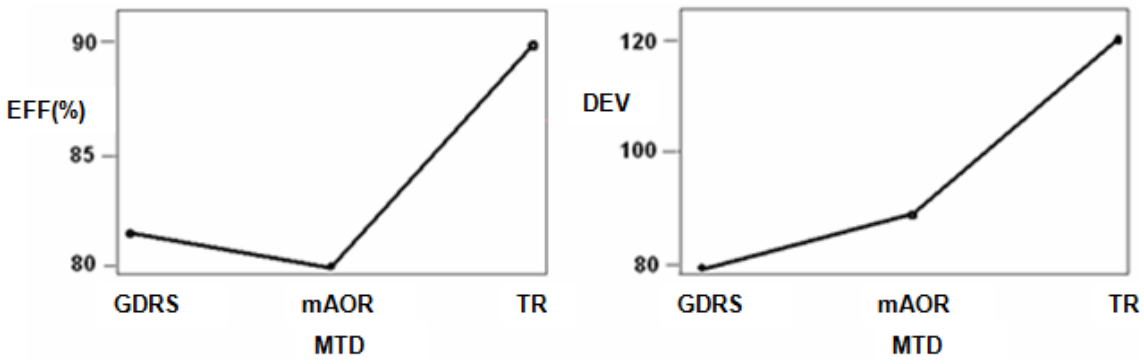


Table 4.4 shows that increasing FLX leads to reactive schedules with significantly better stability. On the other hand, increasing SIZ has a significant negative effect on stability. The MAG factor has a significant interaction effect with MTD on the efficiency and the stability of the revised schedules. While the schedules obtained by both GDRS and mAOR seem to be significantly affected by the increase in MAG, schedules obtained by TR are less affected by such an increase.

4.2.2.4 Urgent existing jobs

Figure 4.18 shows the one-way ANOVA results for the urgent existing job disruption. Results show that the GDRS and TR approaches result in schedules with significantly higher efficiencies than those obtained by mAOR. Moreover, the difference between the efficiencies of TR and GDRS is not significant. Likewise, GDRS and TR result in schedules with significantly lower deviations than those obtained by mAOR, again with no significant difference between the deviations resulting from GDRS and TR. Similar to the arrival of new job disruption, mAOR again results in considerably reduced effectiveness towards both efficiency and stability. The similarity in efficiency and deviation resulting from GDRS and TR for this type of disruption can be attributed to the existence of the urgent job in the original schedule. Furthermore, because the new due date has to be respected, both approaches were forced to re-insert the operations of the urgent job in similar positions on the corresponding machines.

Figure 4.18: One-way ANOVA results for the urgent existing job experiment

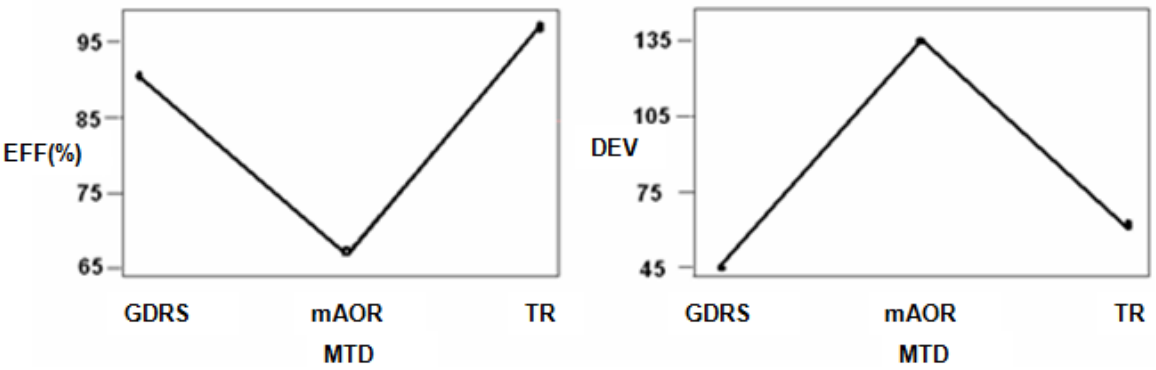


Table 4.4 shows that increasing MAG has a significant negative effect on the efficiency of the revised schedules in general. This is because, with the tighter revised

due date, the three reactive scheduling approaches are forced to schedule the urgent job as the first job in the revised schedules. Furthermore, there is a significant interaction effect between SIZ and MTD on both the efficiency and stability of the schedules. The schedules obtained by the mAOR experience an improvement in efficiency when the problem size gets larger, while the efficiencies of schedules of the other two approaches remain unaffected by such change. As for the stability, while the three approaches suffer from more deviation with the increase in SIZ, GDRS is less affected by such an increase.

4.2.2.5 Order cancellations

Figure 4.19 shows the one-way ANOVA results for the order cancellation experiment. Results show that TR results in schedules with significantly higher efficiencies than GDRS that in turn result in schedules with significantly higher efficiencies than mAOR. As for stability, no significant difference between mAOR and GDRS in deviation of the revised schedules is experienced. However, TR suffers from significantly higher deviations than the former two approaches.

Figure 4.19: One-way ANOVA results for the order cancellation experiment

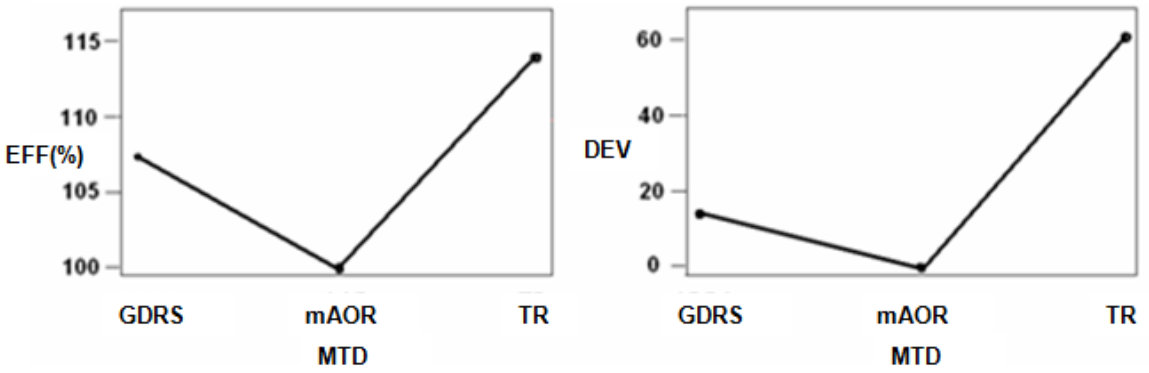
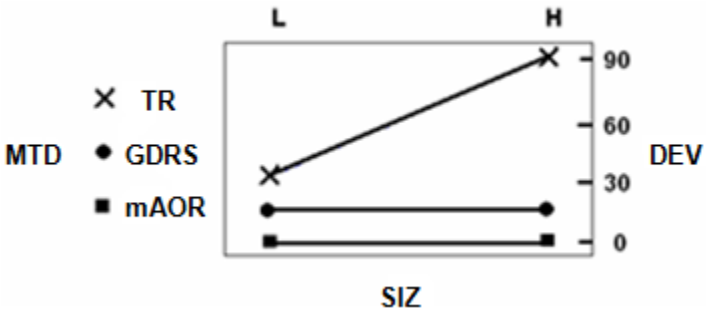


Table 4.4 shows that, the interaction effect between SIZ and MTD on both efficiency and stability is significant. While GDRS tends to maintain the resulting level of deviation with the increase in SIZ, TR is negatively affected by that increases as shown in Figure 4.20. As for the efficiency, the GDRS and TR approaches are both negatively affected by the increase in SIZ. However, note that for this type of disruption, the efficiency and deviation resulting from the mAOR remain constant at almost 100% and zero, respectively.

Figure 4.20: Significant interaction effects for the order cancellation experiment



4.5 Conclusions

In this chapter, the proposed insertion algorithm OI was utilized to perform deadlock-free reactive scheduling for flexible job shops. In Section 4.2, it was shown how OI can be used to apply the job insertion (JI) and the total rescheduling (TR) approaches to revise the production schedules when new jobs are added to the system. In Section 4.3, OI was applied through GDRS to react to other four types of system disruptions; machine breakdowns, process time variations, urgency of existing jobs, and order cancellations. Two experimental studies were also conducted in this chapter; the first (Section 4.2.2) compared the performances of the JI and the TR approaches using three relative performance criteria when new jobs are added to the system, and the second compared

the performances of GDRS, mAOR, and TR when reacting to five different types of system disruptions.

From the first experiment, it can be concluded that the difference in MFT, which was shown to be insignificant in average, does not always justify applying TR over JI. In addition, when an approximate approach, like OI, is utilized to apply TR, the difference in solution time between JI and TR becomes insignificant to justify applying JI all the time. However, the only measure that has shown a significant difference between the applications of the two approaches was the deviation of the revised schedules from the original schedules, which is significantly higher in the TR case.

In view of these findings, two relative practical measures, RNERV and RMFT, have been proposed to assist in choosing between JI and TR. The average values obtained for these two measures were 1.28 for RNERV and 0.1 for RMFT. Multiplying RNERV and RMFT by other system parameters provide, in time units, the expected differences in deviation between TR and JI, and in MFT between JI and TR, respectively. In addition, it has been shown that different types of costs can be combined with each of these measures to provide the decision maker with actual cost figures, upon which the selection of the appropriate approach in a given situation can be based. However, ANOVA results showed that the average values obtained for these measures are not suitable for application in all cases:

- When the processing times and the number of new jobs are considerably large, the threshold values of 3.0 and 0.22 for RNERV and RMFT, respectively, should be considered.

- When a large number of new jobs with small processing times are added to the system, the value of RMFT decreases to only 0.05 with no change in RNERV. This would clearly justify the use of JI over TR in such cases.
- When a buffer space is present, especially in a large system, the value of RMFT increases to 0.21. This combined with the fact that the presence of buffer space had no significant effect on RNERV, could justify the utilization of TR in such cases.
- Having routing flexibility, on one hand decreases RNERV to about 1.0, and on the other also decreases RMFT to about 0.07. Hence, justifying the use of one of the two approaches over the other in such cases is not practical, and cost figures should be estimated.

As for the second experiment, the ANOVA results can be outlined in the following conclusive points:

- GDRS clearly outperformed the mAOR approach in terms of both the efficiency and stability of the revised schedules when reacting to most disruptions.
- GDRS provided revised schedules with higher stability than TR for all types of disruptions, and in most of the cases the differences in stability were significant.
- When reacting to the machine breakdowns, arrival of new jobs, and urgency of existing jobs disruptions, on average, no significant difference in efficiency was observed between the schedules revised using TR and GDRS.
- The effectiveness of GDRS towards the efficiency and stability of the revised schedules tends to be less affected by the increase in either the magnitudes of the disruptions, or the size of the considered problem compared to mAOR and TR.

- TR provided revised schedules with significantly higher efficiency but also with significantly lower stability than those provided by GDRS, when reacting to process time variations and order cancellations.
- Regardless of the utilized approach, having flexibility in the system in terms of alternative routes and buffer spaces enhances the capability of obtaining revised schedules with good efficiency and stability.

It can be noticed that the detailed analysis of the new job arrival disruption conducted in the first experiment, provided some guidelines that can be followed when choosing between TR and JI to react to this type of disruption. It can also be noticed that the ANOVA results of this experiment showed that the average performance measure values should not be followed all the time, because some combinations of values of the system parameters can considerably change the values of these measures, and hence the decisions based on their values. The second experiment was less detailed than the first, since it was conducted on a larger number of disruptions. However, its results have proven the overall efficiency of GDRS over mAOR. Its results further showed that the performance of GDRS is to a great extent comparable to that of TR in reacting to most of the disruptions, with a considerable improvement in the deviation values. Nevertheless, to obtain similar guidelines to the ones provided by the first experiment, more detailed experiments should be conducted to compare the performances of TR and GDRS when reacting to machine breakdowns, process time variations, urgency of existing jobs, and order cancellations.

CHAPTER 5:

Supervision of Automated Manufacturing Cells

5.1 Introduction

In accordance with the adopted control approach (Figure 1.4), after obtaining the deadlock-free scheduling and reactive scheduling tools that constitute the upper level of the control hierarchy (scheduler), the task now is to attain the lower level control module (supervisor) that can transform the schedules into control actions. This module will perform monitoring and dispatching activities that can realize the original and the updated (reactive) instructions coming from the scheduler on the shop floor. Accordingly, the objective of this chapter is to introduce, analyze and verify a formal approach that can transform a given schedule into an *implementable* supervisor. This supervisor should be capable of driving a manufacturing system in the correct (deadlock-free) and optimized manner acquired by the schedule.

It should be noted that the proposed approach can be extended to model controllers for larger systems that may constitute different types of and/or more than one automated material handling devices such as multiple robots and AGVs. However, in this study, focus is limited to obtaining controllers for robotic manufacturing cells, in which all the

material handling tasks between a number of CNC machines are performed by a single robot manipulator. Extending the proposed approach to model controllers for larger systems can be part of a future study.

In Chapter 2, it has been shown that Petri net (PN) supervisors in general can be easily transformed into a programming language interpretable by a programmable logic controller (PLC) or executed through a central computer. It has also been concluded that PN supervisors embedding a marked graph (MG) structure can be easily verified for liveness (deadlock-freeness) and are more suitable for direct implementation since they do not feature any conflicts in their structure. Accordingly, the proposed supervisor generation approach first involves transforming a deadlock-free schedule into a live and reversible MG. Using a hybrid approach earlier proposed in literature [115], through a series of top-down and bottom-up steps, the obtained MG is then augmented with additional places to capture all the events that can take place in the system. These augmentation steps preserve the liveness and reversibility of the initial MG, and eventually provide the required supervisor.

In order to guarantee the liveness of the initial MG, the schedule used to generate this MG must be deadlock-free. In Chapter 3, two deadlock-free scheduling approaches were proposed; the first involved the utilization of MIP models and the second utilized an insertion algorithm (OI). The insertion algorithm was further utilized in Chapter 4 to obtain deadlock-free reactive schedules. The deadlock-freeness of the schedules obtained using the MIP models was preserved by the constraints that prevented a job from holding a machine while waiting for the next machine in its processing route to become available.

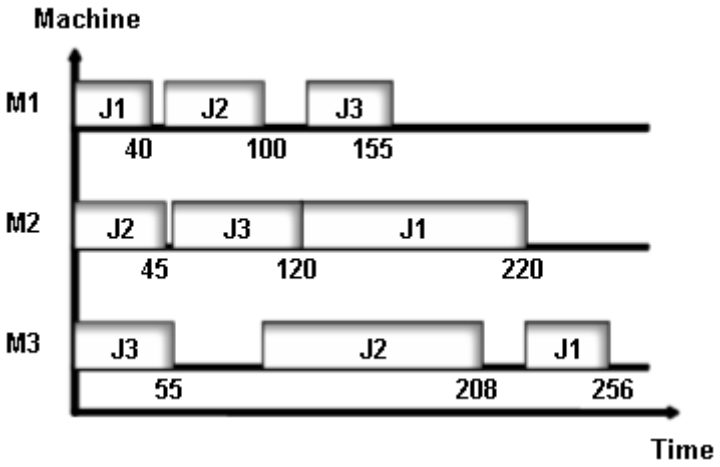
As for OI (Section 3.3.3), two conditions for deadlock occurrence were identified and prevented during the schedule generation process. However, the sufficiency of these conditions for deadlock prevention has yet to be proven.

In this chapter, first the steps required to transform a schedule into an MG are defined (Section 5.2). This will be followed in Section 5.3 by the definition of a necessary and sufficient condition for deadlock occurrence in a schedule. The sufficiency of this condition is proven by analyzing the types of circuits that exist in the MG representing the schedule. In addition, the equivalence of the combination of the two conditions for deadlock defined in Section 3.3.3 to the necessary and sufficient condition will be shown. In Section 5.4, it is shown using an illustrative example and the corresponding MG how that condition can detect a deadlock, and how such a deadlock can be resolved using a buffer. In Section 5.5, the steps of the hybrid approach used to obtain a live and reversible supervisor from the schedule generated MG are illustrated. In Section 5.6, the proposed supervisor generation approach is verified by generating and simulating the supervisors of two benchmark scheduling problems. This is followed by the conclusions of this chapter in Section 5.7.

5.2 Transforming a Schedule into a MG

Consider the schedule of three jobs on three machines shown in Figure 5.1.

Figure 5.1: A schedule of three jobs on three machines

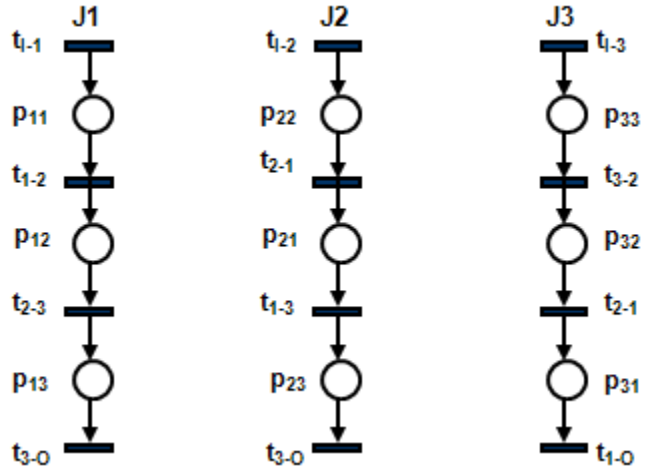


The first step to transform a schedule into a MG is to represent the processing route of each job by a production Petri net (PPN) [30], but without the resource (machine) places associated with each processing operation on the job (Figure 2.2). This reduced PPN provides the sequence of places and transitions that describe the flow of the job through the system. The places represent the required processing operations to produce the job, and the transitions model the release and/or acquisition of the corresponding machine that perform the associated operations. A token in these places, which will be referred to henceforth as *flow places*, indicates that a job is currently holding the corresponding machine, either while begin processed or while waiting for the next machine in its route to become available.

Initially, all the flow places are token-free (empty), indicating that no jobs are present in the system. The processing routes of the jobs shown in Figure 5.1 can then be represented by the reduced PPNs shown in Figure 5.2. In this figure, each flow place p_{ij} represents a job i holding a machine j either for processing or waiting, and each transition

$t_{j\text{-}s}$ represents the release of machine j and acquisition of machine s. Transitions of the type $t_{I\text{-}j}$ and $t_{j\text{-}O}$ model the acquisition of the first machine and the release of the last machine in a job's route, respectively.

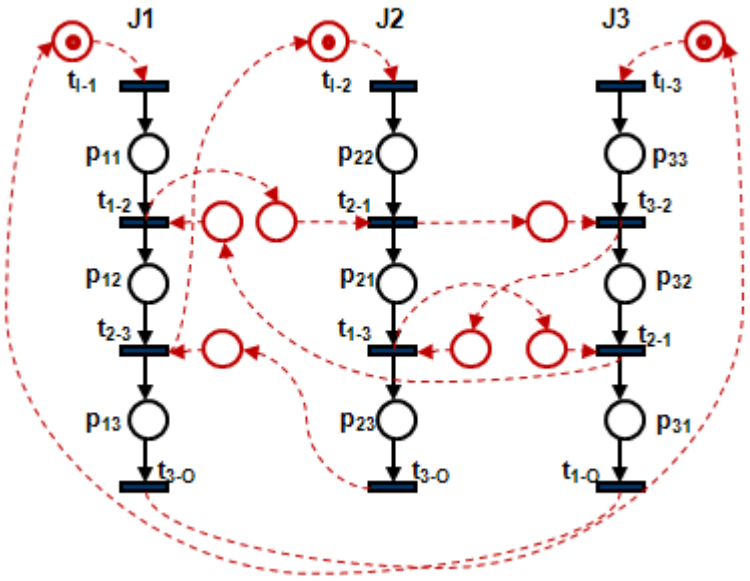
Figure 5.2: Three PPCs of the three jobs



The next step in the transformation process is to represent the sequence of jobs visiting each machine as indicated by the schedule. This is done by connecting each transition representing the release of some machine to an additional place by an input arc. This additional place is then connected to the transition representing the acquisition of the same machine by the next visiting job by an output arc. Accordingly, this latter transition will not be allowed to fire (assign the machine to the next job) before the first transition fires (the machine is released by the previous job). In order to ensure the repetition of the schedule for control purposes (reversibility), a final place is added between the transition that releases the machine from the last job in the visiting sequence, and the transition that acquires the machine for the first job in the sequence. Furthermore, a token is placed in this latter place to allow the initiation of the schedule.

Thus, for each machine, a number of additional places equal to the number of jobs visiting the machine are added to the net. The group of places associated with each machine is included in one circuit, with a single token in the whole circuit. These places will be henceforth referred to as *scheduling places*. Adding these places to the three reduced PPCs shown in Figure 5.2, results in the net shown in Figure 5.3. Note that the resulting net is still a MG.

Figure 5.3: MG of the schedule (SMG)



This representation of a schedule as a MG will henceforth be referred to as a *scheduling marked graph* (SMG). It can be noticed that the SMG captures the first three necessary conditions for deadlock occurrence (Section 1.1.2); no pre-emption, mutual exclusion, and hold while wait. The mutual exclusion and no pre-emption conditions are preserved by the existence of a single token in the circuit associated with each machine. Accordingly, at any given time each machine can be only acquired by a single job, and

the processing of a job on the machine must be completed before any other job can acquire the machine.

As for the hold-while wait condition, since the firing of each transition represents both the release of the current machine and the acquisition of the next machine in a job's route, a job will keep holding a machine until the next machine in its processing route is released. For example in Figure 5.3, the scheduling place connecting $t_{1\text{-}2}$ of J_1's PPC to $t_{2\text{-}1}$ of J_2's PPC ensures that J_2 will keep holding M_2 until M_1 is released by J_1. Consequently, J_3 cannot acquire M_2 until J_2 acquires M_1. When $t_{1\text{-}2}$ of J_1's PPC fires, indicating the release of M_1 by J_1, a token will be placed in its output scheduling place, enabling $t_{2\text{-}1}$ of J_2's PPC to fire whenever p_{22} acquires a token.

It should be noted that similar MG models were used in previous literature to represent production schedules. In [116], a MG model of the schedule was utilized to evaluate the performance regarding minimizing the cycle time. In that approach, a *command* circuit with one token was added for each machine to connect the transitions that corresponded to the *availability and utilization* of the machine. A token in an input *command place* to a transition indicated that the job that corresponds to that transition was the next job to be processed on the machine. Accordingly, the token was initially placed in the input command place to the first job scheduled on the machine, which was also the output command place from the last job scheduled on the machine. However, it should be noted that in that approach, deadlock considerations were not taken into account. Unlike the present model, in that previous model a single transition modeled both the acquisition and the release of the same machine, and its output flow place modeled the waiting of a job

for the next machine in its route. Accordingly, that model did not capture the hold while wait condition, and was prone for deadlocks. In [54], MGs were used to model schedules to verify the absence of deadlocks. A very similar model to the present one was used, and it was concluded that schedules expressed by such models are deadlock-free if and only if every circuit in the MG contained at least one token.

5.3 Deadlock Analysis using MGs

In a MG, each place has exactly one input and one output transitions. Hence, a circuit in a MG is like a siphon (Appendix A). In other words, if a circuit in a MG becomes token-free, no transition firing sequence in the net will be able to place any tokens in this circuit, and all the transitions associated with this circuit become dead. Accordingly, as mentioned earlier, a MG is live if and only if it does not contain any *empty* circuits. Consequently, a schedule is deadlock-free if its corresponding SMG is free of empty circuits.

The simplest procedure to check the liveness of a MG is to first remove any places that contain tokens, along with their input and output arcs, from the net. If the resulting net is circuit-free, then the original MG is live. From Figure 5.3, it can be noticed that the only places that contain tokens in a SMG are the input scheduling places to the transitions associated with the jobs firstly visiting each machine according to the schedule. Removing these places, along with their corresponding arcs, results in a reduced token-free SMG.

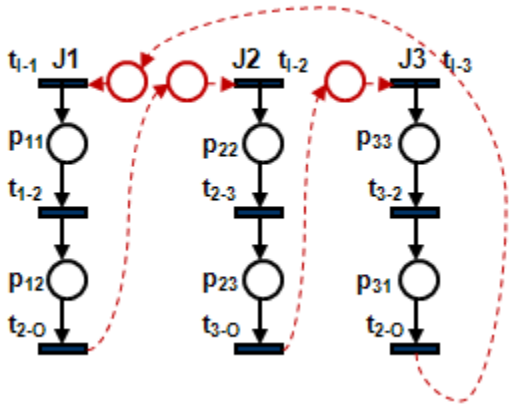
5.3.1 Circuits in a SMG

There are two types of circuits that can exist in a token-free SMG; circuits resulting from cycles or those resulting from deadlocks in the corresponding schedule [75, 117]. A cycle in a schedule is a chain of sequence and flow dependant operations, in which the last operation has to precede the first operation on some machine. A schedule containing a cycle is an *unfeasible schedule* because the sequences of jobs on the machines violate the processing routes of the jobs. For example, Figure 5.4 shows the processing routes of three jobs and their sequences on three machines, as defined by a schedule. This schedule features a cycle because it requires the processing of the second operation of J_3 before the first operation of J_1on M_1. It also requires that the second operation of J_1 precedes the first of J_2 on M_2 and hence the second of J_2 on M_3, which in turn precedes the first operation of J_3 on M_3. Because these requirements violate the processing routes of the jobs, the schedule is unfeasible and its SMG contains an empty circuit ($t_{I\text{-}1} - t_{1\text{-}2} - t_{2\text{-}O} - t_{I\text{-}2} - t_{2\text{-}3} - t_{3\text{-}O} - t_{I\text{-}3} - t_{3\text{-}2} - t_{2\text{-}O} - t_{I\text{-}1}$) as shown in the figure.

Figure 5.4: An infeasible schedule with a cycle

Job	Processing Route
J_1	$M_1 - M_2$
J_2	$M_2 - M_3$
J_3	$M_3 - M_1$

Machine	Job Sequence
M_1	$J_3 - J_1$
M_2	$J_1 - J_2$
M_3	$J_2 - J_3$



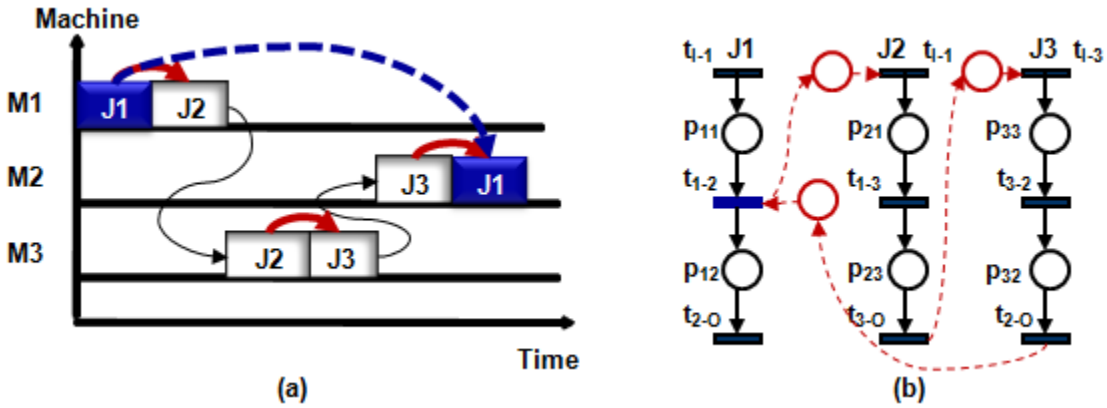
According to the above, the presence of an empty circuit in a SMG of a *feasible* schedule can only be due to a deadlock in that schedule [54]. It should be noted that the MIP models and the insertion algorithm (OI) proposed in Chapter 3 to solve the deadlock-free scheduling problem do not generate schedules containing cycles by default. This is because, while the constraints defined in the former prevent such cycles, the rank matrix representation utilized in the latter does not allow the formation of cycles in the associated schedules [75].

It can be noticed from Figure 5.4 that, because a cycle is a chain of sequence and flow dependant *operations*, each transition associated with the corresponding circuit in the SMG must be either preceded or followed by a flow place in the circuit. This flow place represents an operation associated with the cycle. On the other hand, in order for a deadlock to occur in a schedule, *at least one* job *i* must keep holding some machine *j* indefinitely while waiting for another machine *s*, hence blocking other jobs from acquiring machine *j*. Consequently, at least one transition that belongs to job *i* in the corresponding circuit in the SMG, must be preceded *and* followed by scheduling places in the circuit. The input scheduling place to this transition is associated with the job that precedes job *i* on machine *s*, and the output scheduling place is associated with the job that follows job *i* on machine *j*. Such a transition will henceforth be referred to as a *blocking transition*, and a job associated with such a transition will be referred to as a *blocking job.*

In Figure 5.5 (a), J_1 will keep holding M_1 indefinitely while waiting for M_2 to be released by J_3. Hence, J_2 will not be able to acquire M_1, and the system will end in a

deadlock. Accordingly, in the corresponding SMG shown in Figure 5.5(b), transition $t_{1\text{-}2}$ of J_1's PPC becomes a blocking transition. This is because it is preceded by the scheduling place associated with the release of M_2 by J_3, and is followed by the scheduling place associated with the acquisition of M_1 by J_2 in the circuit ($t_{1\text{-}2} - t_{I\text{-}1} - t_{1\text{-}3} - t_{3\text{-}O} - t_{I\text{-}3} - t_{3\text{-}2} - t_{2\text{-}O} - t_{1\text{-}2}$) .

Figure 5.5: a) A deadlock in a schedule; b) corresponding circuit in SMG



From the above analysis, it can be deduced that a feasible schedule will have a deadlock if and only if its corresponding SMG features a *deadlock circuit*; an empty circuit with at least one blocking transition. In other words, a deadlock circuit in a SMG is a *necessary and sufficient condition* for a deadlock to occur in the corresponding schedule. Note that having more than one blocking transition in a deadlock circuit indicates the existence of a circular wait between more than one blocking jobs. In the figures to follow, scheduling places along with their input and output arcs will sometimes be replaced by bold arcs for ease of illustration.

5.3.2 A Necessary and Sufficient Condition for Deadlock Occurrence

Recall that, a set of jobs is in a circular wait if each job is holding a machine while waiting for the next machine in its processing route, which is in turn held by another job in that same set. Along with the mutual exclusion, no pre-emption, and hold-while-wait conditions, the circular wait condition is traditionally the fourth necessary condition for a deadlock to occur in a job shop (Section 1.1.2). Because a production schedule is set in advance before implementation on the shop floor, it may feature a deadlock that does not literally comply with the circular wait condition with its traditional definition.

From a scheduling perspective, a circular wait could occur between jobs that are already holding some machines in the shop and other jobs that would acquire these machines in the future, as determined by the schedule. For example, in Figure 5.5(a), J_1 will keep holding M_1 indefinitely while waiting for M_2 because the processing of J_3 precedes that of J_1 on M_2, and J_3 cannot acquire M_2 before J_1 releases M_1 according to the defined sequences of J_1, J_2 and J_3 on M_1 and M_3. Hence, there is a circular wait between J_1 and J_3, although at the actual time of occurrence of the deadlock (when J_1 is completed on M_1), J_3 would have not entered the system yet. Accordingly, the circular wait condition should be extended to account for such situations in a schedule.

A deadlock circuit in a SMG can be viewed as a sequence of transitions in which at least one transition (blocking transition), belonging to a job (blocking job), is directly preceded and followed by transitions that belong to a different job (or jobs) in the circuit. As mentioned earlier, each of these transitions model the release of a machine and/or the acquisition of another. From a scheduling perspective, each transition represents the

completion of an operation and/or the beginning of the next operation in a job's processing route. Accordingly, a deadlock circuit in a SMG can be interpreted in the corresponding schedule as a sequence of *pairs of directly consecutive* operations in routes of jobs, which starts and ends with an operation pair of some job *i*. The operation pairs that precede and follow job *i*'s operation pair in the sequence must belong to jobs other than *i*. The relation between each two consecutive operation pairs in the sequence must either follow the processing requirements (job routes) or the sequencing requirements defined by the schedule. Such a sequence of operation pairs will henceforth be referred to as a *circular block*, and can be formally defined as follows:

Definition: Let $\sigma_j(o_{ik})$ be the processing order of the k^{th} operation of job i on some machine j in the schedule. Let $O = \{(o_{ik}, o_{i(k+1)})\}$, be the set of ordered pairs of operations corresponding to precedence relations in routes of all jobs and $Q = \{(o_{ik}, o_{pr})\}$ such that $\sigma_j(o_{pr}) - \sigma_j(o_{ik}) = 1$, be the set of ordered pairs of operations that correspond to the processing order on all the machines as defined by the schedule. A ***circular block*** exists in a production schedule if and only if for some $(o_{ik}, o_{i\,(k+1)}) \in O$:

$$\boldsymbol{\sigma(o_{ik}) < \sigma(o_{pr}, o_{p(r+1)}) < \ldots < \sigma(o_{qw}, o_{q(w+1)}) < \sigma(o_{i\,(k+1)})}$$

where:

a. $\sigma(o_{ik}) < \sigma(o_{pr}, o_{p(r+1)})$ if (o_{ik}, o_{pr}) or $(o_{ik}, o_{p(r+1)}) \in Q$.

b. $\sigma(o_{pr}, o_{p(r+1)}) < \sigma(o_{qw}, o_{q(w+1)})$ if:

i. (o_{pr}, o_{qw}) or $(o_{pr}, o_{q(w+1)}) \in Q$, or

ii. $p = q$ and $r + 1 = w$, or

iii. $(o_{p(r+1)}, o_{qw})$ or $(o_{p(r+1)}, o_{q(w+1)}) \in Q$ where $o_{p(r+1)}$ is the last operation of job p.

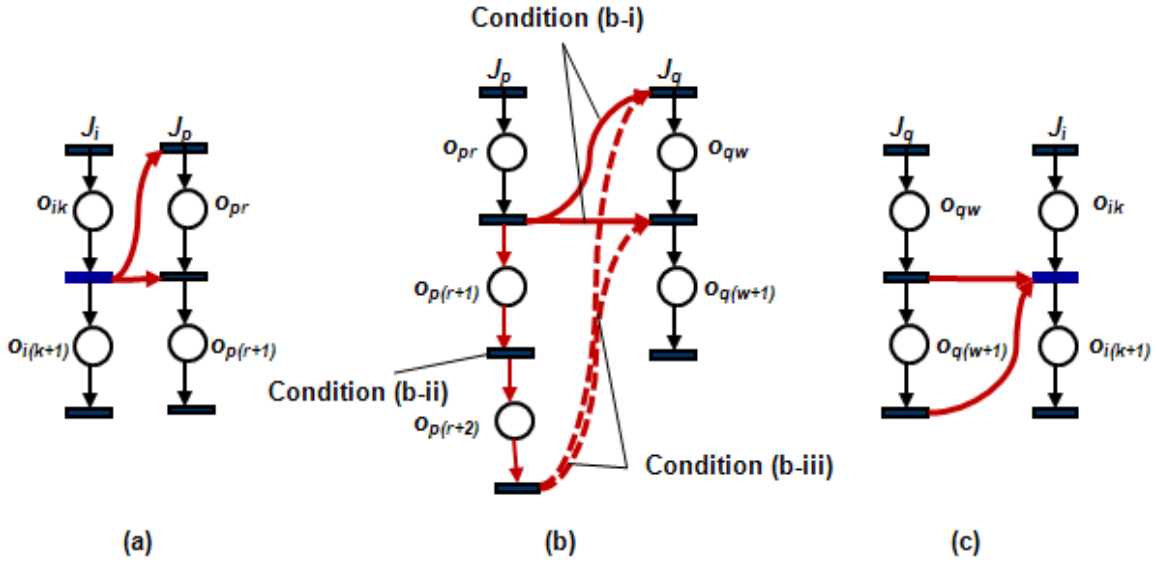
c. $\sigma(o_{qw}, o_{q(w+1)}) < \sigma(o_{i\,(k+1)}) \leftarrow (o_{qw}, o_{i(k+1)})$ or $(o_{q(w+1)}, o_{i(k+1)}) \in Q$.

Condition (a) indicates that an operation o_{ik} precedes an operation pair $(o_{pr}, o_{p(r+1)})$ in a circular block if either the processing of o_{pr} or $o_{p(r+1)}$ follows the processing of o_{ik} on some machine.

Condition (b) indicates that an operation pair $(o_{pr}, o_{p(r+1)})$ precedes another operation pair $(o_{qw}, o_{q(w+1)})$ in a circular block if: i) either the processing of o_{qw} or $o_{q(w+1)}$ follows the processing of o_{pr} on a machine, ii) both operation pairs belong to the same job and $(r+1)$ is equivalent to w, or iii) the processing of $o_{p(r+1)}$, which is the last operation of job p, precedes that of o_{qw} or $o_{q(w+1)}$ on a machine. Stated in plain words, when an operations pair $(o_{pr}, o_{p(r+1)})$ is reached when trailing a circular block, potentially two new pairs of operations are considered in the next step. The first is the pair that includes the operation that follows o_{pr} on the corresponding machine. The second is $((o_{p(r+1)}, o_{p(r+2)})$, which is the next operation pair in job p's processing route. However, if $o_{p(r+1)}$ is the last operation in job p's processing route, the pair that includes the operation that follows $o_{p(r+1)}$ on the corresponding machine is considered.

Finally, condition (c) completes the circular block if the processing of o_{qw} or $o_{q(w+1)}$ precedes the processing of $o_{i(k+1)}$ on some machine. In the corresponding SMG, these conditions can be interpreted as shown in Figure 5.6 (in this figure, places are associated with the corresponding operations for illustration).

Figure 5.6: Illustration of the conditions associated with a circular block: a) condition (a); b) condition (b); c) condition (c)

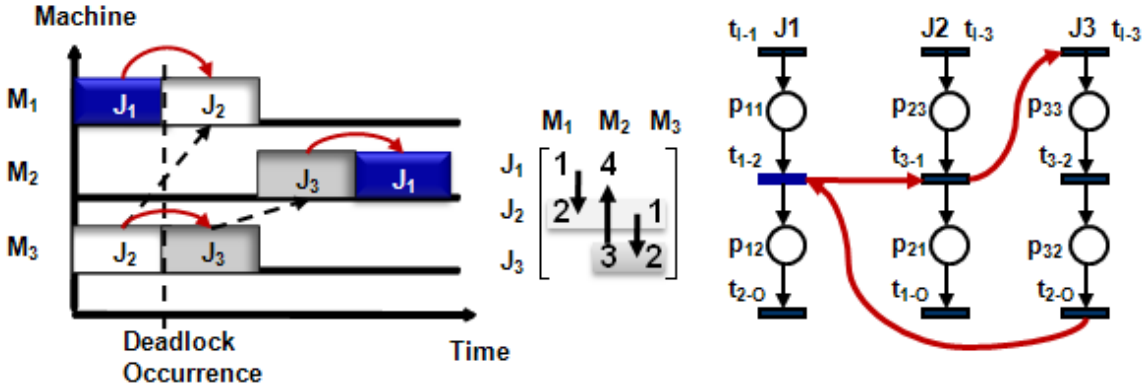


From Figures 5.6 (a) and (c), it can be seen that conditions (a) and (c) mark the beginning and the end of the deadlock circuit at the blocking transition; condition (a) necessitates that the output of this transition be a scheduling place, and condition (c) necessitates the same for its input place. Condition (b) trails the rest of the circuit by: i) moving from the current job *p* to a new job *q* through a scheduling place, *and* ii) moving from the current pair of operations to the next pair in the current job's route, *or* iii) moving from the last operation in the current job's route ($o_{p(r+2)}$) to a new job through a scheduling place. Accordingly, condition (b) can trail the circuit from and back to the blocking transition through any possible arc in the net. Thus, the conditions associated with the circular block definition are sufficient to track any circuit in a SMG that features at least one blocking transition. Hence, it can be deduced that a circular block in a schedule is *a necessary and sufficient* condition for the occurrence of a deadlock.

5.3.3 Interpretation in the Rank Matrix

Recall from Chapter 3 that the value of position p_{ij} in a rank matrix that corresponds to the k^{th} operation of job i, provides in column j the processing order of job i on machine j, and in row i the order of operation o_{ik} in job i's route. Accordingly, the rank matrix captures both sets O and Q that define the processing routes of jobs, and their precedence relations on the machines, respectively. Furthermore, the value of $\sigma_j(o_{ik})$ is equivalent to the value of position p_{ij} with regards to column j. Thus, a circular block can be determined using the rank matrix of a schedule by comparing the values of the rank matrix positions using the above defined conditions of a circular block. For example, re-consider the schedule that was shown earlier in Figure 3.10, along with its rank matrix and associated SMG as shown in Figure 5.7.

Figure 5.7: Illustration of a circular block using a rank matrix



The deadlock in this schedule can be deduced using the circular block condition and the associated rank matrix as follows:

- $(o_{11}, o_{12}) \in O$, where o_{11} is processed on M_1 and o_{12} is processed on M_2. This can be easily deduced from the rank matrix since $p_{11} = 1$ and $p_{12} = 4$.

- Condition (a): $\sigma(o_{11}) < \sigma(o_{21}, o_{22})$ because $(o_{11,} o_{22}) \in Q$ on M_1 (Note that o_{22} is the second operation of J_2 that is performed on M_1). In the rank matrix, this can be deduced by comparing the values of p_{11} and p_{21}, which indicates that the value of p_{11} is less than that of p_{21}.

- Condition (b-i): $\sigma(o_{21}, o_{22}) < \sigma(o_{31}, o_{32})$ because $(o_{21,} o_{31}) \in Q$ on M_3. In the rank matrix, p_{23} (o_{21}) is less than p_{33} (o_{31}).

- Condition (b-iii): $\sigma(o_{31}, o_{32}) < \sigma(o_{11}, o_{12})$ because $(o_{32}, o_{12}) \in Q$ on M_2 and o_{32} is the last operation of J_3. In the rank matrix, p_{32} (o_{32}) is less than p_{12} (o_{12}).

- Condition (c): this condition is satisfied in the last step because o_{12} is reached, and hence the circular block condition is complete.

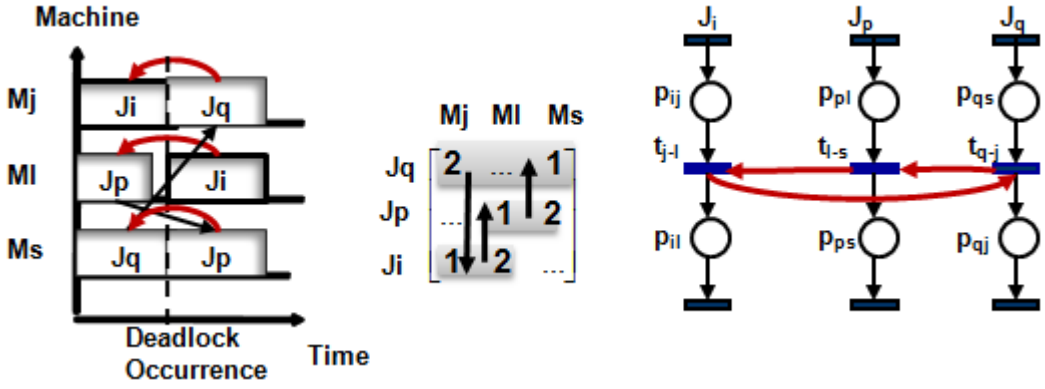
Therefore $\sigma(o_{11}) < \sigma(o_{21}, o_{22}) < \sigma(o_{31}, o_{32}) < \sigma(o_{12})$, and hence the above schedule features a deadlock that will occur when J_1 completes processing on M_1. It should be noted that J_2 is also a blocking job in this deadlock, and the circular block could have also been deduced by starting with o_{21} instead of o_{11} (note that $t_{3\text{-}1}$ in the PPC of J_2 is a blocking transition).

5.3.4 Sufficiency of the Pre-defined Deadlock Conditions

Combining the two conditions for deadlock occurrence defined in Section 3.3.3 results in the circular block condition. The first of these two conditions used the rank matrix of the schedule to recognize any circular wait in the given schedule. In fact, the sub-algorithm proposed to recognize these circular waits using the rank matrix (Sub-

algorithm 2) follows conditions (a), (b-i), and then (c) of a circular block. Thus, it does not recognize deadlocks that may result from proceeding through conditions (b-ii) and (b-iii). It is only capable of recognizing a circular wait, with its traditional definition, that is caused by a set of jobs holding a set of machines as shown in Figure 5.8.

Figure 5.8: Circular wait recognition using Sub-algorithm 2

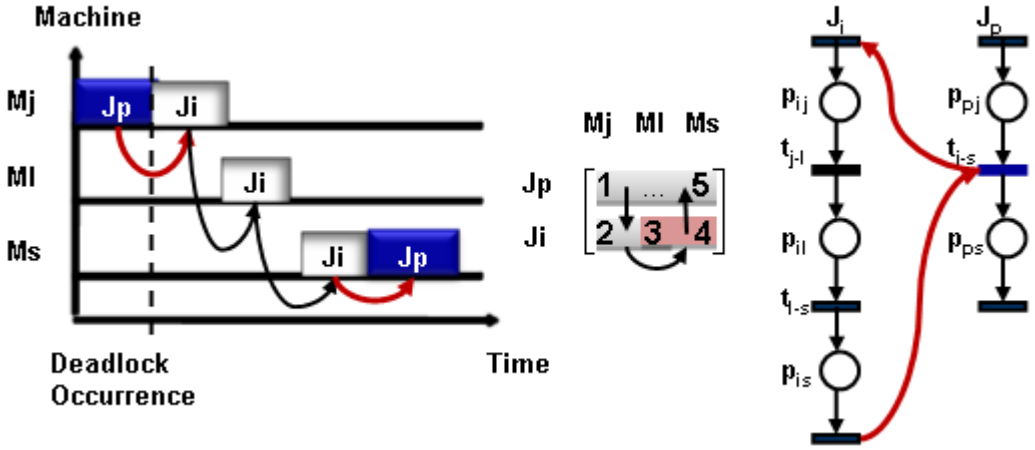


It can be noticed from the SMG in Figure 5.8 that, a deadlock circuit associated with a circular wait constitutes only blocking transitions, and accordingly all the jobs involved in this circuit are blocking jobs. Indeed, such a circuit does not pass by any flow places since each of the involved jobs is *currently* holding a machine while waiting for another machine.

The second condition that recognizes what has been referred to as unfeasible sequence using the rank matrix in Sub-algorithm 3, follows conditions (a), (b-ii) or (b-iiii), and then (c) of a circular block. In other words, it recognizes any job whose operations are *contained within* two directly consecutive operations of another job as shown in Figure 5.9. From the rank matrix shown in Figure 5.9, it can be seen that the deadlock is tracked

from position p_{pj} to position pair (p_{ij}, p_{il}) using condition (a), then from position (p_{ij}, p_{il}) to (p_{il}, p_{is}) using condition (b-ii), and finally from (p_{il}, p_{is}) to (p_{pj}, p_{ps}) using condition (b-iii) or from (p_{il}, p_{is}) to p_{ps} using condition (c). It can also be noticed that the circuit in the SMG features only one blocking transition.

Figure 5.9: Unfeasible sequence recognition using Sub-algorithm 3



From the above analysis, it can be concluded that the circular block condition is a combination of the traditional circular wait condition, and the unfeasible sequence condition that results from jobs involved in circular waits that have not yet acquired their respective machines. Combining both conditions through the circular block condition is sufficient to recognize any deadlock in the schedule resulting from a circular wait (Figure 5.8), an unfeasible sequence (Figure 5.9), or a complex circular wait like the one shown in Figure 5.7.

5.4 Deadlock Detection and Resolution

When there is no buffer space in the system, a schedule that features a circular block will eventually drive the system into a deadlock. Consequently, in the analysis of such systems, it is sufficient to detect the presence of a circular block in a schedule to reject it. However, in the case of the existence of some buffer space, such circular blocks can be resolved by scheduling some job to reside in the buffer after the completion of an operation. Accordingly, analysis of this case should further involve recognizing the jobs that could resolve the blocks by residing in the buffer.

5.4.1 Complexity of Detection and Resolution of Circular Blocks

Many approaches have been proposed in literature to detect circuits in MGs. One of these approaches is the Topological Ordering Algorithm (TOA) that was proposed in [118]. This algorithm was originally proposed to obtain a topological order of a group of nodes forming a network. It starts with determining the nodes with no input arcs, and eliminates their output arcs that are inputs to other nodes. It keeps repeating this procedure on every new node that ends with no input arcs. If the algorithm terminates with nodes that still have input arcs, this indicated the existence of a circuit in the network. The possibility of using this algorithm to detect circuits in MGs was mentioned in [117], by considering the places and transitions of a MG to be the nodes of a network. In the case of SMGs, the TOA can be employed by considering only the transitions as the nodes of the network.

The advantage of using the TOA over other algorithms is that, it has a worst case complexity (number of steps) equal to the number of nodes of the network. However, the TOA can only detect but not resolve deadlocks. Hence it can only be employed in systems that do not feature a buffer space. OI, on the other hand, can detect and resolve deadlocks with the same worst case complexity. For each evaluation of a position of an operation in the schedule, the algorithm in the worst case performs a number of steps equal to the number of operation pairs in the routes of jobs already inserted in the schedule. This corresponds to the total number of transitions in the corresponding SMG less the number of input and output transitions to and from the system. However, it should be noted, that resolving the deadlocks using the buffer requires detecting all the circular blocks that involve the operation being inserted, which usually require more steps than just detecting the existence of a circular block. Fortunately, the number of steps required to detect all the blocks is still limited by the mentioned worst case complexity (this will be illustrated in Section 5.4.4).

5.4.2 Reducing the Search Space

In accordance with the search pattern of the circular block condition that tracks operation pairs in routes of jobs, an initialization routine can be performed to reduce the search space involved in the process. It resembles to a great extent the TOA, but is applied on operation pairs with no successors (or transitions with no output arcs) rather than those with no predecessors. This is because pairs of operations with no predecessors cannot be reached when trailing a circular block, and hence there is no need to eliminate

them from the search space since they cannot be reached in the first place. On the other hand, eliminating operation pairs (transitions) with no successors can reduce the search space, and will ensure that eventually the ones remaining in the schedule (SMG), are members of circular blocks (deadlock circuits). This initialization routine can be added to Step 3.2 in OI and be performed as follows:

Initialization routine: Eliminating Operations with No Successors:

Step 1. Set $E = \{\} \leftarrow$ set of operations that can be eliminated, $e = 0 \leftarrow$ indicator of operations that can be eliminated.

Step 2. $\forall o_{ik} \in P_j$ (processed on machine j) such that o_{ik} is the last operation of J_i that is not an element of E, if $\forall\ o_{pr} \in P_j$ such that $o_{pr} \notin E$, $p_{pj} < p_{ij}$, then $E \leftarrow E \cup o_{ik}$, $e \leftarrow 1$.

Step 3. $\forall o_{ik} \in P_j$ such that $k = 1$ and $o_{i(k+1)} \in E\ \forall k$, if $\forall\ o_{pr} \in P_j$ such that $o_{pr} \notin E$, $p_{pj} < p_{ij}$, then $E \leftarrow E \cup p_{ij}$, $e \leftarrow 1$.

Step 4. If $e = 1$, $e \leftarrow 0$, Go to Step 2. Else, STOP.

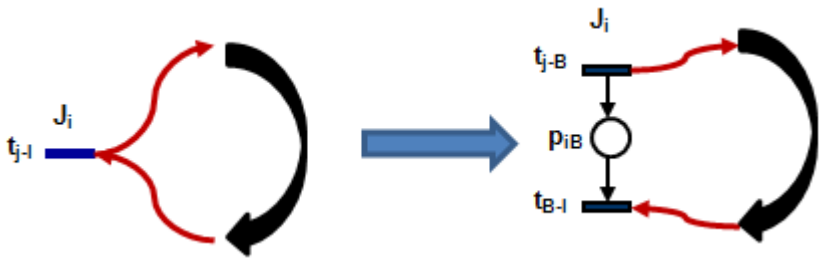
Step 2 of the routine adds to E any operation, which is not a member of E, that has no successor operation on the associated machine and in its corresponding job's route. In Step 3, if all the operations of a job, except for its first operation, have been added to E and this first operation has no successor on its associated machine, then this operation is also added to E. The routine is used in Section 5.4.4 to reduce the search space of an insertion step in an illustrative example.

It should be noted that, adding an operation to the set E is equivalent to eliminating the corresponding flow place in the SMG, with all its input and output arcs. However, adding an operation to E does not mean that this operation will not be considered in the search process, with its predecessor operation in the job's route, in an operation pair. Only when its predecessor is as well added to E, an operation will not be considered in the search process in an operation pair. This is equivalent to eliminating a transition from the SMG, whose input and output flow places are the ones corresponding to the eliminated operation pair.

5.4.3 Resolving a Circular Block

Recall from Section 5.3.1 that a blocking job in a SMG is one that features a blocking transition in a deadlock circuit. In the corresponding schedule, a blocking job is the one whose associated operation pair is both preceded and followed by operation pairs that belong to other jobs in a circular block. Such job will keep holding a machine indefinitely while waiting for the next machine in its route to become available, which causes a deadlock. Hence, if such a job resides in the buffer after completing processing on the potentially blocked machine, the corresponding circular block will be resolved. In the corresponding SMG, this is equivalent to expanding the blocking transition into a flow place $p_{i\mathrm{B}}$ in-between two transitions, $t_{j\text{-}B}$ and $t_{B\text{-}l}$, as shown in Figure 5.10.

Figure 5.10: Resolving circular blocks

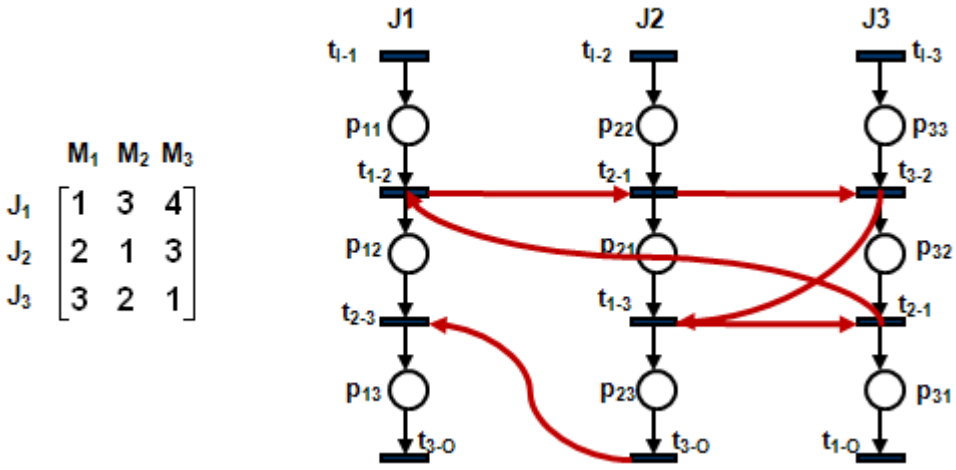


A token in p_{iB} represents the blocking job while residing in the buffer. Firing $t_{j\text{-}B}$ releases the potentially blocked machine j and places job i in the buffer, while firing $t_{B\text{-}l}$ acquires machine l and moves job i from the buffer. It can be noticed from Figure 5.10 that this setting eliminates the potential circuit in the SMG.

5.4.4 Illustrative Example

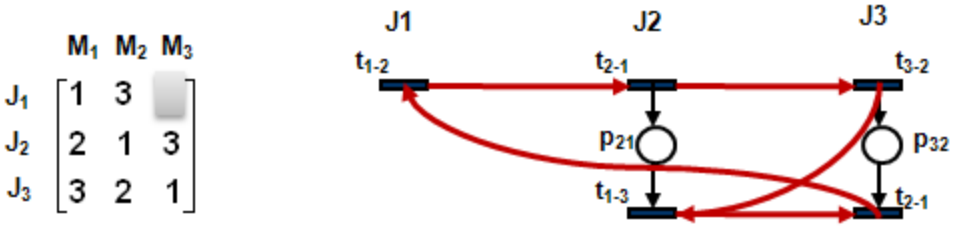
Consider the schedule shown earlier in Figure 5.1 and its associated SMG shown in Figure 5.3. Furthermore, consider that the operation currently being inserted in the schedule by OI is o_{33}, the third operation of J_3 that is processed on M_1, and that the currently considered position for this operation is the third on M_1. Figure 5.11 shows the rank matrix of this schedule, along with its corresponding SMG after removing the token-occupied scheduling places that mark the beginning of the processing on the machines (recall that deadlocks in SMGs are only caused by token-free circuits).

Figure 5.11: Rank matrix and SMG of illustrative example



By applying the initialization routine, the third operation of J_1 on M_3, and the third operation of J_3 on M_1 are firstly added to *E*, since they have no successors. Consequently, o_{23} on M_3 and o_{12} on M_2 are added to *E*, and eventually $E = \{o_{13}, o_{33}, o_{12}, o_{23}\}$. Accordingly, operation pair (o_{12}, o_{13}) can be eliminated from the search space, and flow places p_{12}, p_{13}, p_{23}, and p_{31} along with transitions $t_{3\text{-}O}$ in J_1's PPC, $t_{3\text{-}O}$ in J_2's PPC, $t_{1\text{-}O}$ in J_3's PPC, and $t_{2\text{-}3}$ in J_1's PPC can be eliminated from the SMG. Since transitions and places with no input arcs cannot be reached in the search process, transitions $t_{I\text{-}1}$ in J_1's PPC, $t_{I\text{-}2}$ in J_2's PPC, and $t_{I\text{-}3}$ in J_3's PPC, and flow places p_{11}, p_{22}, and p_{33} can also be eliminated from the SMG. The rank matrix and SMG can then be reduced as shown in Figure 5.12.

Figure 5.12: Reduced rank matrix and SMG of illustrative example



Since the operation currently being inserted in the schedule is o_{33} on M_1, the search process for circular blocks starts with the operation pair (o_{32}, o_{33}) on machines M_2 and M_1, respectively, which corresponds to transition $t_{2\text{-}1}$ in J_3's PPC. Following the circular block definition, three circular blocks (circuits) can be found for this example as shown in Figure 5.13 and Table 5.1:

Figure 5.13: Search steps for circular blocks

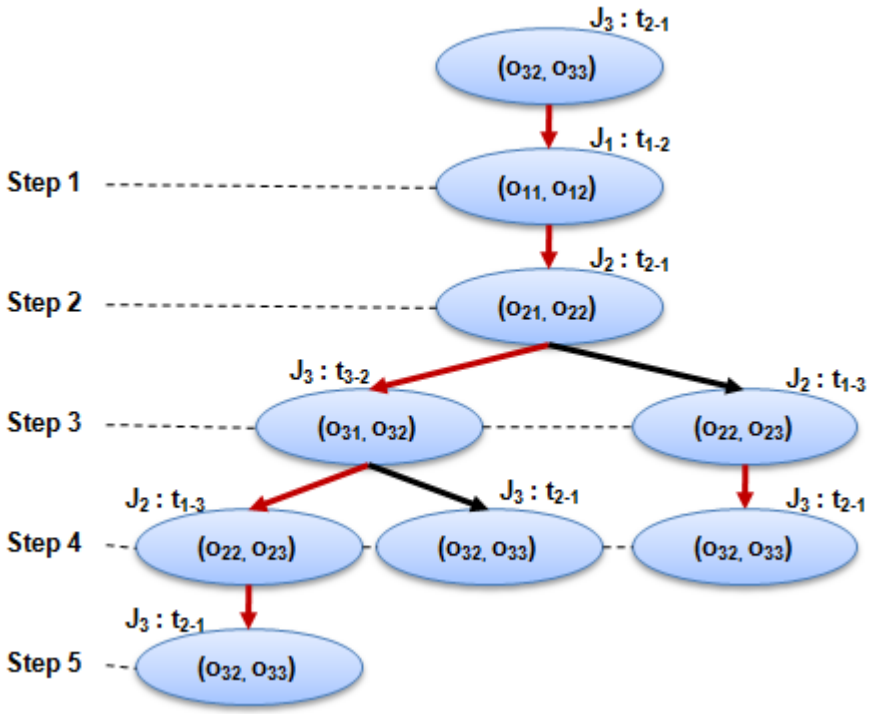


Table 5.1: Circular blocks (circuits) of illustrative example

Circular Block 1	Circular Block 2	Circular Block 3
(o_{32}, o_{33})	(o_{32}, o_{33})	**(o_{32}, o_{33})**
(o_{11}, o_{12})	**(o_{11}, o_{12})**	**(o_{11}, o_{12})**
(o_{21}, o_{22})	**(o_{21}, o_{22})**	**(o_{21}, o_{22})**
(o_{22}, o_{23})	(o_{31}, o_{32})	**(o_{31}, o_{32})**
		(o_{22}, o_{23})

From Figure 5.13, it can be noticed that there are three circular blocks in this example because two operation pairs, (o_{21}, o_{22}) and (o_{31}, o_{32}) each had two successor operation pairs. It should also be noted that only five search steps were required to determine all the circular blocks in the schedule, which is equal to the number of operation pairs in the reduced rank matrix and the number of transitions in the reduced SMG. Furthermore, if the system had no buffer capacity, it would have only required four steps (circular blocks 1 and 2) to determine that the considered schedule features a deadlock that cannot be resolved.

Assuming that the considered system features some buffer capacity, the job(s) that can be used to resolve the three circular blocks, by being placed in the buffer, must be identified. Blocking jobs can be determined from the identified circular blocks, as the jobs whose operation pairs are preceded and followed by operation pairs of other jobs in the circular blocks. Accordingly, from Table 5.1 the blocking jobs can be determined as follows:

- Block 1: J_3-($o_{32,}$ o_{33}); J_1-($o_{11,}$ o_{12}).
- Block 2: J_1-($o_{11,}$ o_{12}); J_2-($o_{21,}$ o_{22}).
- Block 3: J_3-($o_{32,}$ o_{33}); J_1-($o_{11,}$ o_{12}); J_2-($o_{21,}$ o_{22}); J_3-($o_{31,}$ o_{32}); J_2-($o_{31,}$ o_{32}).

In order to ensure the best utilization of the available buffer space, blocking jobs whose associated operation pairs feature the maximum number of appearances in the circular blocks are set to reside in the buffer. This is because, when a blocking job is placed in the buffer, its associated circular block is resolved (Section 5.4.3). Accordingly, for the considered example, since operation pair J_1: ($o_{11,}$ o_{12}) appears in the three blocks,

J_1 is selected to reside in the buffer after completing its processing on M_1. This is sufficient to resolve the three circular blocks in the schedule.

5.5 Realizing the Supervisor from the Schedule

After obtaining the best deadlock-free schedule for a system, a supervisor could be obtained. This supervisor will not only ensure the correct behavior of the system like conventional supervisory controllers, but will further guarantee the optimized performance obtained by the schedule. The following procedure outlines the required steps to transform a SMG of a schedule into a supervisor. The procedure follows the *hybrid synthesis approach* proposed in [115]. The objective of this approach is to start with a simple abstract model of a system in which important features, like liveness and reversibility, can be easily or have already been verified. Then, through a number of refinement (top-down) and aggregation (bottom-up) steps that preserve the liveness and reversibility of the initial model, a complete model that provides the necessary details to fully describe the system can be obtained.

The proposed procedure starts with the SMG of the problem on hand. It should be noted however, that in the case of systems with a buffer space, the SMG is not circuit-free until the buffer places are added to the SMG. This is followed by adding the previously omitted scheduling places that contain the tokens, after which the SMG becomes live and reversible by design. Finally, the necessary nodes required to represent the material handling (robot) tasks and to organize the utilization of the buffer space are

added to the SMG. This finally results in an augmented SMG (ASMG) that can realize the control actions required to drive the system.

Procedure: Realizing a Supervisor from the SMG:

The supervisor realization procedure constitutes four main steps:

1. Adding buffer places and transitions to the SMG.
2. Re-inserting the token-occupied scheduling places.
3. Adding material handling places and transitions.
4. Organizing the utilization of the buffer capacity by adding command circuits.

The steps of the procedure can be illustrated using the illustrative example from Section 5.4.4 as follows:

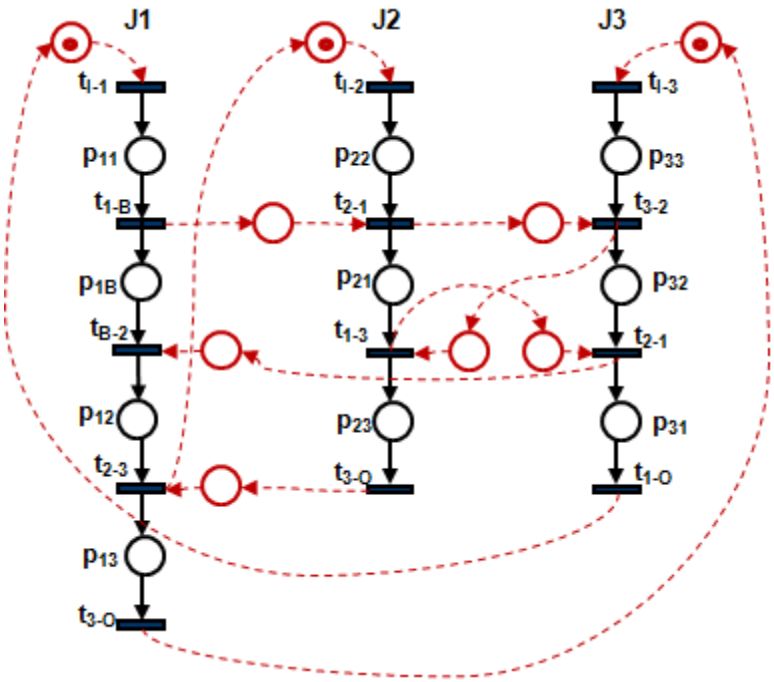
Step 1. Adding buffer places and transitions:

The best schedule found using OI for the problem on hand is the one shown in Figures 5.1 and 5.11 (assuming the existence of a unit capacity buffer). In the previous section it was realized that if J_1 resides in the buffer after completion of processing on M_1, this schedule will be deadlock-free. Accordingly, a buffer place p_{1B} along with two transitions, $t_{1\text{-}B}$ and $t_{B\text{-}2}$, are added to the PPC of J_1 to replace the blocking transition $t_{1\text{-}2}$. This, as previously shown in sections 5.4.3 and 5.4.4 eliminates the three circuits in the SMG, which becomes circuit-free.

Step 2. Re-inserting the token occupied scheduling places:

Recall that at the beginning of the deadlock analysis (Section 5.3), the token-occupied scheduling places, which mark the beginning of processing on each machine, were removed to detect token-free circuits in the SMG. Accordingly, up to this point, the SMG of the schedule cannot be live since it does not contain any tokens. By re-inserting these places with their corresponding arcs in the circuit-free SMG, each place in the net becomes included in a circuit that contains at least one token. Hence, the SMG becomes live, and since it still retains its marked graph structure, it also becomes reversible [119]. The live and reversible SMG of the illustrative example can then be obtained after adding the buffer place and transitions in J_1's PPC and inserting the token-occupied scheduling places as shown in Figure 5.14.

Figure 5.14: Live and reversible SMG of illustrative example



Step 3. Adding material handling (robot) place and tasks:

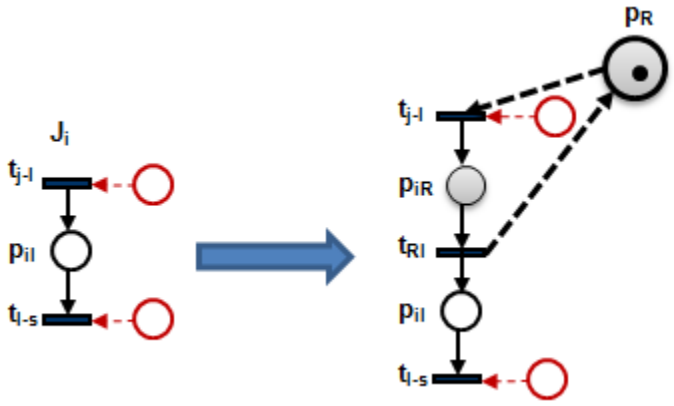
This step involves top-down decomposition and bottom-up aggregation sub-steps. These two sub-steps do not affect the liveness or the reversibility of the SMG [115]. The top-down decomposition involves place refinement of the *flow* places of the PPC of each job to model the robot acquisition and release tasks. Since each flow place models a processing operation on a machine, in practice it is associated with a robot task that delivers the corresponding job to the acquired machine. Hence, each flow place is decomposed into two places with a transition in-between. The first place models the job while being handled by the robot, while the second preserves the function of the original flow place.

In order to ensure that the robot is not acquired simultaneously by more than one material handling function, a bottom-up aggregation step adds a robot place p_R with one token to the SMG. This place is connected with output arcs to every transition that models the acquisition of the robot, and input arcs from every transition that models the release of the robot. This ensures that when the robot is acquired to perform some task, p_R becomes token-free, and hence the robot cannot be acquired to perform another task. The top-down decomposition and bottom-up aggregation steps are shown in Figure 5.15.

In Figure 5.15, transition $t_{j\text{-}l}$ models the acquisition of the robot to transfer job i from machine j to machine l and the acquisition of machine l. Hence, $t_{j\text{-}l}$ cannot fire unless place p_R contains a token (indicating that the robot has been released), and when it fires, the token in p_R is removed. It should be noted that $t_{j\text{-}l}$ remains the output transition of the scheduling place that enables J_i to acquire M_l and accordingly cannot be fired until this

place acquires a token. Place p_{iR} models the handling operation of J_i by the robot, and transition t_{Rl} models the delivery and the initiation of processing of Ji on M_l. Accordingly, firing t_{Rl} returns the token to p_R, and the robot becomes available again.

Figure 5.15: Addition of robot place and tasks to the SMG



According to this setting, a robot cannot be acquired to transfer a job to a machine until this machine becomes available to process this job. This ensures the deadlock-free operation of the system when the supervisor is implemented, since the robot can never deliver a job to an already occupied machine. It should also be noted that adding the robot place p_R, along with its input and output arcs, negates the marked graph structure of the SMG. Hence an SMG after adding the robot place will be referred to as an *augmented scheduling marked graph* (ASMG).

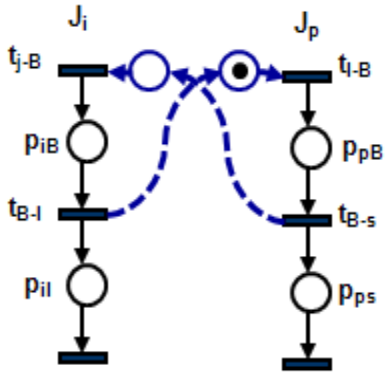
Step 4. Organizing the utilization of the buffer capacity:

The utilization of the available buffer space is more complicated than that of the robot since it is not only sufficient to ensure that the capacity of the buffer will not be violated, but also that the jobs visit the buffer in a correct order. For example, assume that two jobs

i and *p* are scheduled to visit the buffer (with a unit capacity) after completing processing on machines *j* and *l*, respectively, and that they are both ready for that move. Further assume that the next machine in J_i's route is M_l, and that it is scheduled to visit this machine after J_p. In this case, if J_i is placed in the buffer before J_p, it will keep holding the buffer indefinitely while waiting for J_p to release M_l, which is in turn waiting for J_i to release the acquired buffer space, and a deadlock will occur. Hence, J_p should visit the buffer before J_i in order to avoid such a deadlock.

Accordingly, in order to realize the correct order by which the buffer is visited, a command circuit, similar to the ones defined for the machines, has to be defined for the buffer space. This can be achieved by adding to the ASMG a circuit containing a number of buffer scheduling places equal to the number of visits made to the buffer, such that only a single token is permitted to circulate between these places. Each buffer place has one output arc to the transition that assigns the buffer to a job, and one input arc from the transition that releases the buffer from the previous job as shown in Figure 5.16. Like the machine command circuits, the circulating token is initially placed in the buffer scheduling place preceding the transition that assigns the buffer to the first visiting job.

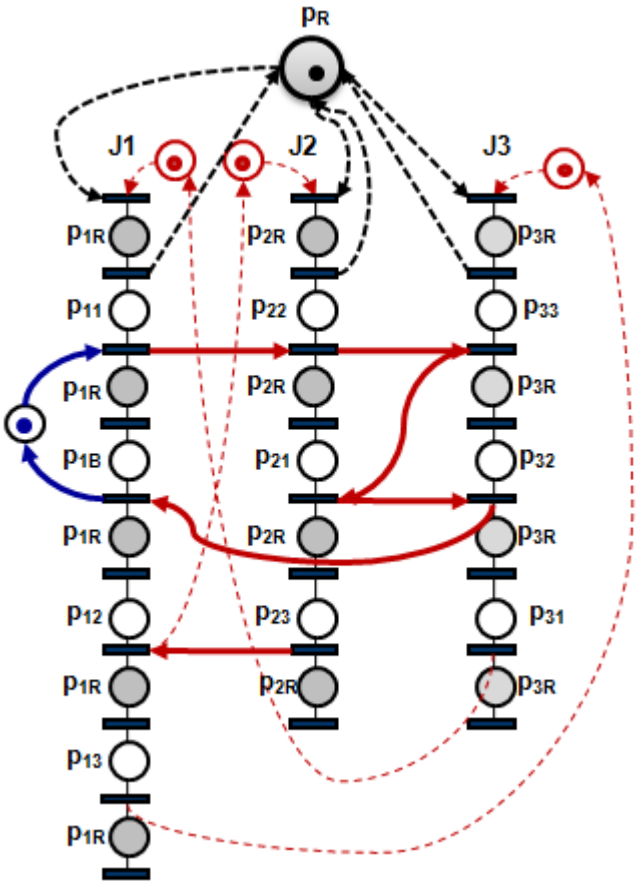
Figure 5.16: Regulating buffer capacity



It should be noted that the order of jobs to visit the buffer is determined from the deadlock-free schedule of the problem. Since the final schedule provides the times at which each job will acquire and release the buffer, in a manner that does not violate the capacity of the buffer, these times can be used to sort the jobs and determine a buffer visiting order that does not violate the deadlock-freeness of the system. It should also be noted that if a system features a buffer capacity greater than one, a command circuit is defined for each unit of the capacity. In other words, the buffer is assumed to have a number of slots equal to its capacity, and for each of these slots a job visiting sequence, and consequently a command circuit, is defined.

After applying steps 3 and 4 of the supervisor realization procedure, the final ASMG can be obtained for the illustrative example as shown in Figure 5.17. In this figure, the arcs that connect p_R to its associated transitions have been omitted for the sake of clarity, with the exception of the set of arcs associated with the robot tasks that transfer the jobs from the system input buffer(s) to the first machines in their corresponding routes. In addition, except for the token-occupied scheduling places, all scheduling places along with their corresponding arcs are represented by bold arcs. It should be noted that the shown ASMG only features one buffer scheduling place because the buffer is only visited once by J_1 after completing processing on M_1.

Figure 5.17: Controller ASMG for the illustrative example



5.6 Simulation and Verification

In order to verify that the supervisor realization approach produces supervisors that can be executed in a deadlock-free manner to complete a set of given jobs, the supervisors for two benchmark problems are generated and simulated in this section. The simulation process entails executing the corresponding controller ASMGs of the problems to simulate the production process. Executing a net means firing all the enabled transitions as they become enabled (Appendix A). If the net completes execution with all the jobs completed, this will indicate that the supervisor can correctly drive the system to

completion upon implementation. However, if the net completes execution with only a fraction of the jobs completed, this will indicate the existence of a local deadlock somewhere in the network. Moreover, if at some point during the execution none of the transitions in the net becomes enabled, this will indicate a total deadlock (all machines or jobs are blocked).

In order to test the reversibility and repeatability of the supervisors, they will be run for lot sizes of five parts for each job type. This can be achieved by adding a place p_{iI} at the beginning of each PPC of a job i in the net, with a number of tokens equal to five, and a place p_{iO} at the end of each PPC to count the number of tokens that will eventually go through, which represents the number of completed parts of each job type.

5.6.1 Selected Problems

The supervisor realization procedure is used to obtain the supervisors of the benchmark problems '4J x 3M' and ft06 that were solved using OI in Section 3.6.2. The processing routes and times for these problems are shown in Tables C.3 and C.4 in Appendix C, respectively. The best objective function values obtained for these problems using OI are shown in Table 3.3. The instance selected for problem '4J x 3M' is the one that featured a unit buffer capacity, with the objective of minimizing the makespan (MS). As for problem ft06, as was shown in Table 3.3, no buffer space is available in the system, and the best MS solution shown in the table is selected. The best MS schedule

obtained along with its associated rank matrices and the corresponding ASMG for problem '4J x 3M' are shown in Figures 5.18 and 5.19, respectively.

Figure 5.18: Schedule and rank matrix of problem '4J X 3M'

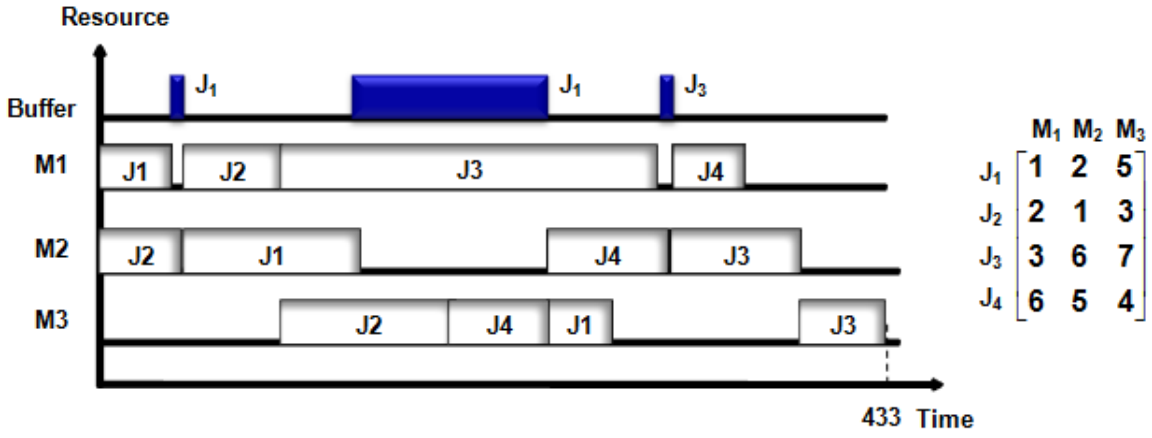
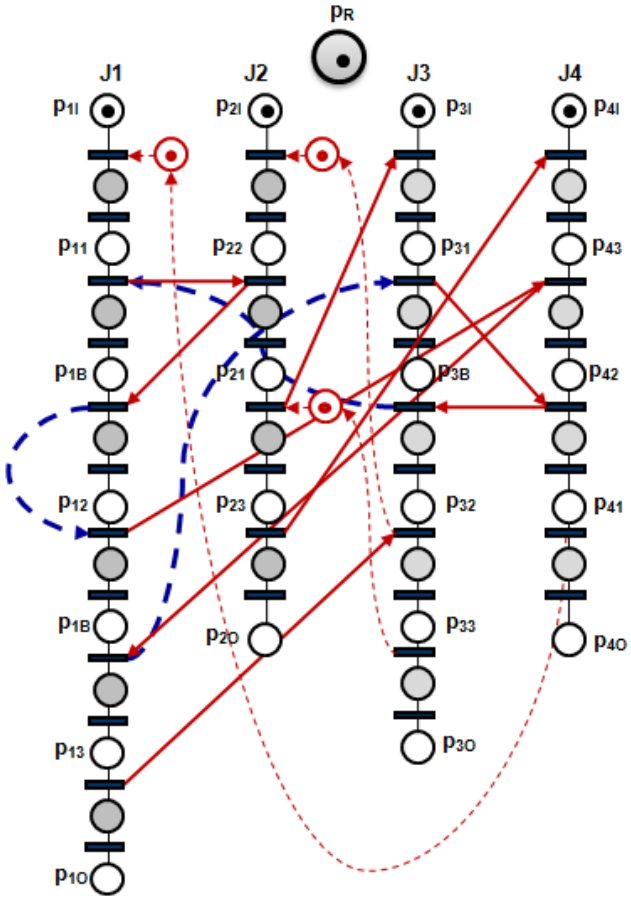
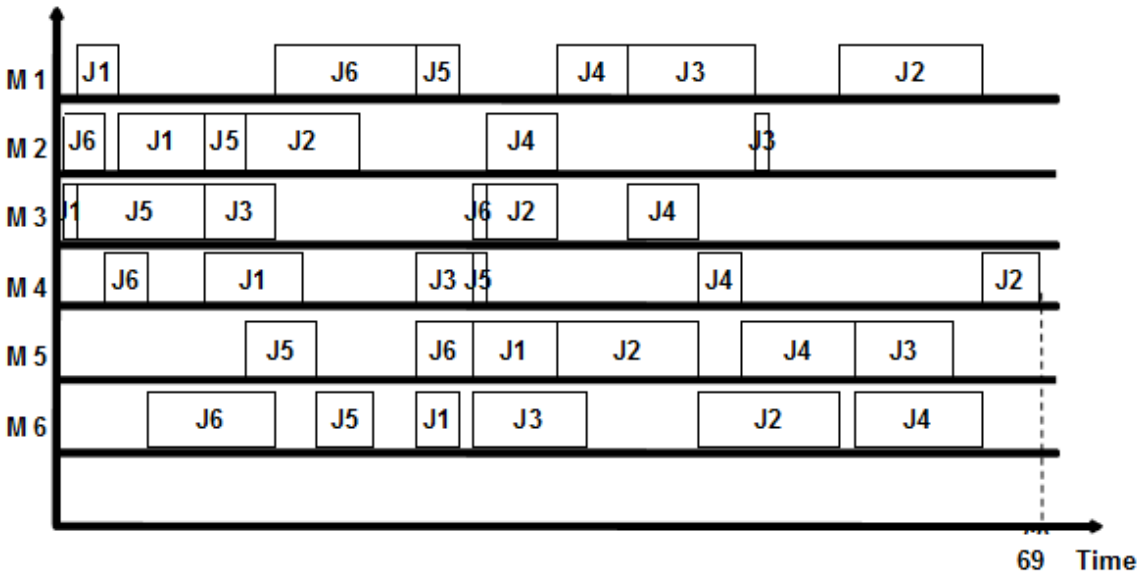


Figure 5.19: Controller ASMG of problem '4J X 3M'



The best schedule and corresponding ASMG obtained for problem ft06 are shown in Figures 5.20 and 5.21, respectively. It should be noted that the ASMGs were obtained for problems '4J X 3M' and ft06 in 0.05 seconds each.

Figure 5.20: Schedule and rank matrix of problem ft06

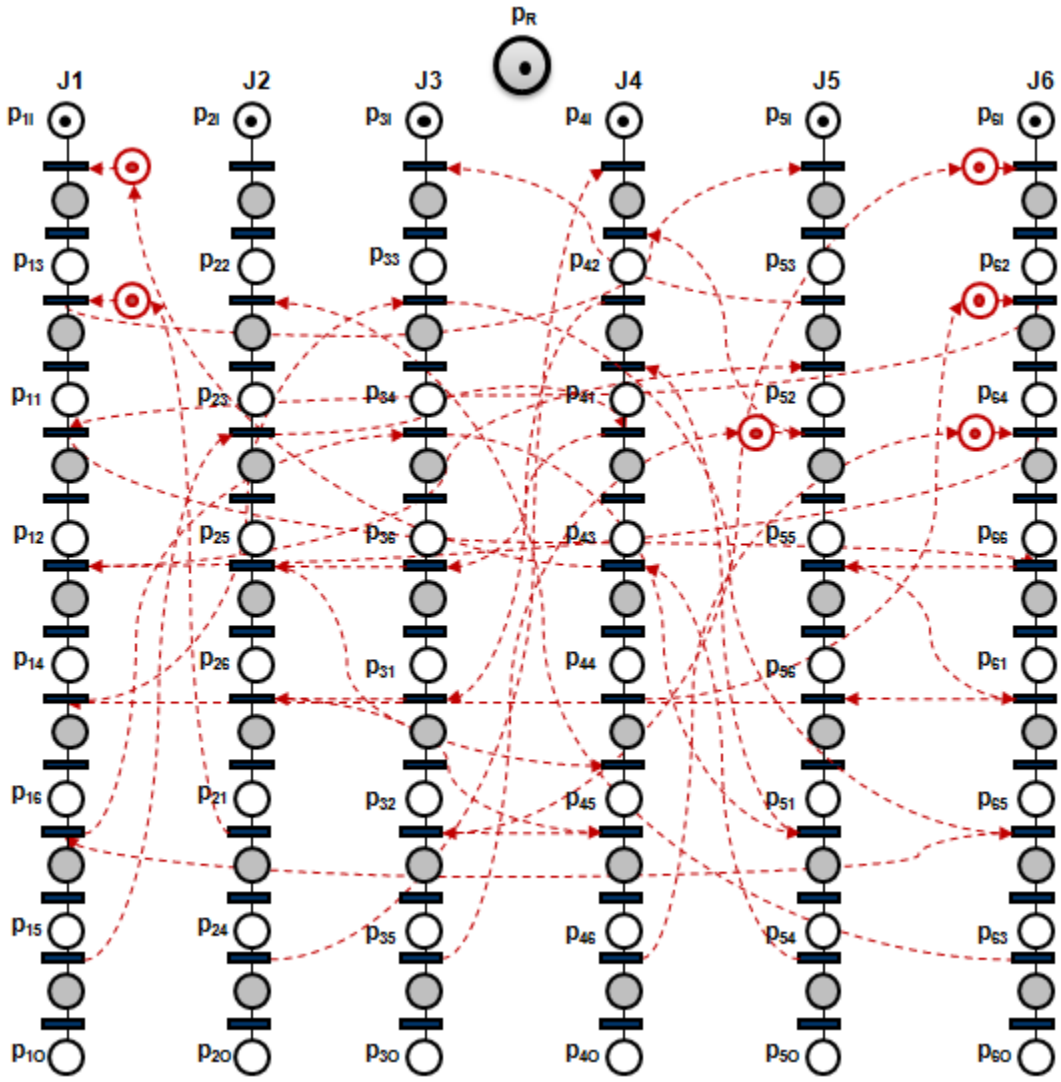


	M_1	M_2	M_3	M_4	M_5	M_6
J_1	2	3	1	4	8	7
J_2	11	5	8	12	9	10
J_3	9	10	3	5	12	8
J_4	8	6	9	10	11	12
J_5	7	4	2	8	5	6
J_6	4	1	7	2	6	3

The two ASMGs were executed, and the five tokens in each p_{iI} place in the two nets reached the corresponding p_{iO} place, indicating that all the parts for all the job types for the two problems were completed successfully. It should be noted that only problems '4J x 3M' and ft06, were selected for the verification process in order to be able to illustrate the obtained controller ASMGs. In fact, the supervisor realization approach was verified

with a variety of problems, some of which were randomly generated and others were from the literature. However, it was not practical to show an ASMG for a 10J X 10M or a 20J X 20M problems.

Figure 5.21: Controller ASMG of problem ft06



5.7 Conclusions

In this chapter, a systematic procedure for transforming a deadlock-free schedule into a supervisor for automated job shop systems was proposed. The overall approach starts by generating a MG that captures the processing routes of the jobs and the sequence of jobs to visit each machine in the system. This MG structure has been referred to as a scheduling marked graph (SMG). The next step was to apply a hybrid synthesis procedure to decompose and augment the SMG to obtain an augmented SMG (ASMG) that can realize all the control actions required to drive a system. The supervisor realization approach was verified by obtaining and simulating the controller ASMGs for two benchmark problems in literature. Simulation results showed that both ASMGs were successfully run to complete all the jobs of the two problems.

In this chapter, a condition for deadlock occurrence in job shop schedules was also proposed. In addition, it was shown using an analysis of the types of circuits that may exist in a SMG that this condition, the circular block condition, is necessary and sufficient to detect any embedded deadlock in a given schedule. Furthermore, it was proven that the combination of the two conditions for deadlock detection presented in Chapter 3 is equivalent to the circular block condition, and that the combination of these two conditions through OI ensures the deadlock-freeness of the obtained schedules. Moreover, an initialization routine was also presented to reduce the search space of the deadlock detection phase in OI, and through an illustrative example, it was shown that the maximum number of steps in this deadlock detection phase is limited by the number of operation pairs in the schedule.

The supervisor realization procedure proposed in this chapter can be applied with any deadlock-free scheduling approach. In other words, the only input to this procedure is a deadlock-free schedule, regardless of the type of approach used to obtain this schedule. Hence, it can be used with the MIP models or the insertion algorithm (OI) proposed in Chapter 3, or any other available deadlock-free scheduling approach or approaches that will be developed in future. Furthermore, focus in this study is to obtain controllers for robotic flexible manufacturing cells. That is why the controller realization approach was more oriented towards robotic tasks and a single material handler availability supposition. Nevertheless, the proposed approach can be extended to model larger and more sophisticated systems that feature more than one material handler, or different material handling types (like robotic manipulators and automated guided vehicles) working synchronously and autonomously. Yet, this remains a direction for future research.

CHAPTER 6:

Implementation in an Experimental FMC

6.1 Introduction

In Chapter 5, it has been shown how a deadlock-free schedule can be transformed into a readily implementable supervisor. This proposed supervisor is represented by a PN structure in which an augmented scheduling MG (ASMG) is embedded. In this chapter, it will be shown how this proposed supervisor can be implemented and utilized to supervise and autonomously drive an experimental flexible manufacturing cell (FMC) to produce different product mixes in a deadlock-free manner. The main objectives of this real-world implementation of the proposed control architecture are to validate the correctness of the scheduler/supervisor levels of the hierarchy and to identify any hardware or software limitations (if any) that may be discovered during its implementation in a real industrial setting.

It was shown in Section 2.5 that a number of approaches can be followed to implement a PN controller in a manufacturing cell. The two main techniques to reach this goal is either to interpret the PN model at run time using an industrial computer, or to transform the PN model using a compiler into a PLC program. It was also mentioned that the

former approach is more robust to sudden changes in the system, and can be used to define rules for resolving conflicts or concurrency situations in the PN controller. Accordingly, this former approach is adopted in this work.

In order to implement the PN controller using a computer, an interface must exist between the computer and the system components to translate the control actions from the PN to the cell components and the feedback from the cell to the PN into digital signals. The digital signals must be interpretable by both levels of the control hierarchy. Thus, the controlled experimental FMC will include three main elements:

i. A personal computer (PC); this constitutes the upper level controller (scheduler) and the lower level controller (supervisor).
ii. The FMC components; these include a robot arm and a number of experimental (virtual) machines and buffers.
iii. A digital Input/Output (I/O) data acquisition module to act as an interface between the computer and the cell components.

In Section 6.2, a detailed description of the designed experimental FMC is presented. In this section, it is also shown how the robot arm is programmed to perform all the required movements in the cell. The task was made more challenging due to the limited number of inputs and outputs on the robot controller. This is followed in Section 6.3 by an illustration of how the execution of the ASMG can be conditioned by feedback signals and translated into control actions to execute the processing tasks in the FMC. In Section 6.4, the results of executing a number of randomly generated product mixes in the

experimental FMC are presented. The conclusions of this chapter are finally outlined in Section 6.5.

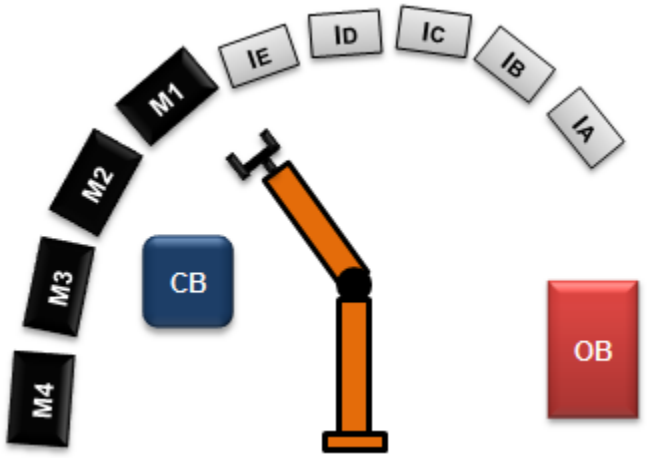
6.2 The Experimental FMC

The experimental FMC is designed and set-up in the Robotics & Automation Laboratory at the University of Manitoba. The designed cell consists of a five degrees of freedom ASEA robot arm, *four* virtual machines, *five* input buffers (I_A – I_E) for five different job types, a central buffer (CB) with a *unit* capacity, and an output buffer (OB) that acts as a system output. To validate the correctness (deadlock-free operation) of the proposed control system, the need for actual machining operations, and consequently actual machines, becomes redundant. It is only required to ensure that all the jobs in the system can follow their processing routes through the system to completion without encountering any deadlock situations. Accordingly, in the experimental FMC, a machine is virtually represented by a physical location that can hold a job and a sensor that provides the status of the machine as being available or acquired by a job. Likewise, the central buffer is represented by a physical location and a sensor.

As mentioned earlier in Chapter 3, the number of machines that can exist in a typical robotic FMC is limited. This is because, along with the different types of buffers that may exist in a cell, the work envelope of the robot arm and the number of pick-up/drop-off (P/D) locations that it can physically and feasibly serve, limit the number of machines that can exist in the cell. Hence, the designed cell contains only four machines, and along

with the five input buffers, the central buffer, and the output buffer, contains in total eleven P/D locations for the robot arm to serve. The experimental FMC is illustrated in Figure 6.1.

Figure 6.1: Experimental FMC in the Robotics & Automation Laboratory at University of Manitoba



6.2.1 Programming the Robot Arm

The ASEA robot arm has five degrees of freedom, and a control interface with 16 input and 16 output lines. The role of the robot arm in the cell is to perform all the material handling operations between all the cell locations (machines and buffers), which also include the P/D operations at each location. In the adopted job-shop manufacturing environment, each job type can have a different processing route through the system. Accordingly, the material handler operating in such systems must be capable of performing all the possible handling operations between any two locations in the cell. It should be noted however, that some movements between locations in the cell are

redundant; these include the movements from any input buffer to the central buffer or the output buffer, from any machine to any input buffer, from the central buffer to the output buffer, and from the output buffer to any other location in the cell. Thus, for a cell employing m machines to process n job types, provided that the robot arm returns to a pre-defined home position after each handling operation, the required movements of the material handler include:

- m movements from each of the n input buffers to each machine,
- one movement from each of the m machines to each of the other m - 1 machines,
- one movement from each of the m machines to the central buffer,
- one movement from the central buffer to each of the m machines, and
- one movement from each of the m machines to the output buffer.

Accordingly, the total number of required handling operations for a n x m job-shop cell with a unit capacity central buffer is: $mn + m(m - 1) + m + m + m =$ $\boldsymbol{(m + n + 2)\, m.}$ Thus, the number of required handling operations by the robot arm in the 5 x 4 experimental FMC is 44 handling operations.

Because the ASEA robot arm provides only 16 input control lines, and since it is required to execute 44 different handling operations, it was not feasible to associate each input line with one handling operation. Accordingly, a combinatorial control input scheme to the robot arm was developed. In this scheme, a combination of a number of input signals to the robot arm is used to execute each of the 44 handling operations. Because the input signals are binary (0-low or 1-high), the maximum number of input

combinations that can be obtained using x input lines is 2^x. Hence, to attain the 44 combinations required to execute the required handling operations, a minimum of six input lines must be utilized to control the robot arm. The used combinations of the values of these lines, and the associated handling operations are shown in Table 6.1.

In addition to the six input lines to the robot arm, a single output line is also utilized to relay the status of the robot arm, as a feedback signal, to the control system. As will be shown later in Section 6.3, this feedback signal indicates whether or not the robot arm is at its home position. In accordance with the above defined control scheme of the robot arm, one main program and 44 sub-programs are developed in the robot arm controller to execute the 44 handling operations. The structures of the main program and one of the sub-programs are shown in Appendix D.

6.2.2 The I/O Data Acquisition Module

As mentioned earlier, in order to translate the action commands and the feedback signals between the computer based controller and the cell components, an interface must be utilized. In the experimental FMC, a Data Translation DT9835™ digital I/O data acquisition module is used. This module has 64 configurable digital I/O lines and 32 dedicated input lines, all in banks of eight lines. In other words, the configurable 64 lines have to be dedicated as input or output in banks of eight lines. The module is connected to the PC externally via a USB cable, and to the cell components using a screw terminal panel.

Table 6.1: Control input scheme for the robot arm

Combination Number	Input Lines						Handling Operation (Movement)
	1	2	3	4	5	6	
1	1	0	0	0	0	0	From Input Buffer 1 to Machine 1
2	0	1	0	0	0	0	From Input Buffer 1 to Machine 2
3	1	1	0	0	0	0	From Input Buffer 1 to Machine 3
4	0	0	1	0	0	0	From Input Buffer 1 to Machine 4
5	1	0	1	0	0	0	From Input Buffer 2 to Machine 1
6	0	1	1	0	0	0	From Input Buffer 2 to Machine 2
7	1	1	1	0	0	0	From Input Buffer 2 to Machine 3
8	0	0	0	1	0	0	From Input Buffer 2 to Machine 4
9	1	0	0	1	0	0	From Input Buffer 3 to Machine 1
10	0	1	0	1	0	0	From Input Buffer 3 to Machine 2
11	1	1	0	1	0	0	From Input Buffer 3 to Machine 3
12	0	0	1	1	0	0	From Input Buffer 3 to Machine 4
13	1	0	1	1	0	0	From Input Buffer 4 to Machine 1
14	0	1	1	1	0	0	From Input Buffer 4 to Machine 2
15	1	1	1	1	0	0	From Input Buffer 4 to Machine 3
16	0	0	0	0	1	0	From Input Buffer 4 to Machine 4
17	1	0	0	0	1	0	From Input Buffer 5 to Machine 1
18	0	1	0	0	1	0	From Input Buffer 5 to Machine 2
19	1	1	0	0	1	0	From Input Buffer 5 to Machine 3
20	0	0	1	0	1	0	From Input Buffer 5 to Machine 4
21	1	0	1	0	1	0	From Machine 1 to Machine 2
22	0	1	1	0	1	0	From Machine 1 to Machine 3
23	1	1	1	0	1	0	From Machine 1 to Machine 4
24	0	0	0	1	1	0	From Machine 1 to Central Buffer
25	1	0	0	1	1	0	From Machine 2 to Machine 1
26	0	1	0	1	1	0	From Machine 2 to Machine 3
27	1	1	0	1	1	0	From Machine 2 to Machine 4
28	0	0	1	1	1	0	From Machine 2 to Central Buffer
29	1	0	1	1	1	0	From Machine 3 to Machine 1
30	0	1	1	1	1	0	From Machine 3 to Machine 2
31	1	1	1	1	1	0	From Machine 3 to Machine 4
32	0	0	0	0	0	1	From Machine 3 to Central Buffer
33	1	0	0	0	0	1	From Machine 4 to Machine 1
34	0	1	0	0	0	1	From Machine 4 to Machine 2
35	1	1	0	0	0	1	From Machine 4 to Machine 3
36	0	0	1	0	0	1	From Machine 4 to Central Buffer
37	1	0	1	0	0	1	From Central Buffer to Machine 1
38	0	1	1	0	0	1	From Central Buffer to Machine 2
39	1	1	1	0	0	1	From Central Buffer to Machine 3
40	0	0	0	1	0	1	From Central Buffer to Machine 4
41	1	0	0	1	0	1	From Machine 1 to Output Buffer
42	0	1	0	1	0	1	From Machine 2 to Output Buffer
43	1	1	0	1	0	1	From Machine 3 to Output Buffer
44	0	0	1	1	0	1	From Machine 4 to Output Buffer

The DT9835 comes with Data acquisition (DAQ) Adapter for MATLAB, which provides an interface between the MATLAB Data Acquisition Toolbox[5] and Data Translation's DT-Open Layers Architecture™. Using the adapter and the toolbox, the module and its I/O lines can be defined and operated by utilizing a series of command functions in the MATLAB command window. Table 6.2 shows the main functions that are used and their purposes:

Table 6.2: Command functions of the I/O module

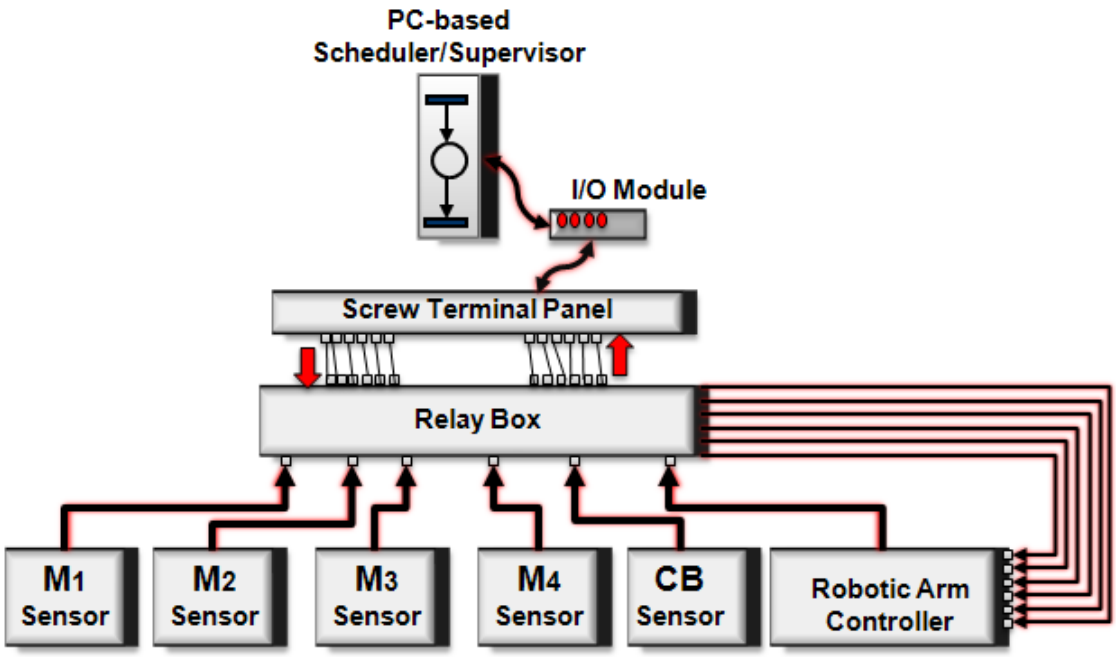
Function	Purpose
digitalio	Creates the I/O module in the MATLAB environment
addline	Adds digital Input or Output lines to the module
getvalue	Reads values (0-1) from the Input lines
putvalue	Writes values (0-1) to the output lines

As mentioned earlier, the machines in the experimental FMC and the central buffer are represented by a physical location that can hold a job, and a digital sensor. Hence, these cell components do not require any input (control action) from the cell controller. Alternatively, they provide feedback signals to the controller indicating their availability status. Accordingly, each of these components is connected through an input line to the I/O module. Along with the output line from the robot arm, the total number of input lines to the I/O module is six; one for each cell component to provide its status. As for the output lines from the I/O module, only six output lines are required to provide the aforementioned 44 input combinations required to control the robot arm.

[5] Copyright 1984 – 2005, The MathWorks Inc.

It should also be mentioned that, the input/output module within the robot controller requires 24 volts and the I/O module generates 5 volts only. Consequently, a relay box is established between the screw terminal panel of the I/O module and the other cell components. The function of the relays is to transform the electrical signals flowing between the I/O module and the robot arm controller to the appropriate voltage values. The overall control architecture for the experimental FMC can be illustrated as shown in Figure 6.2.

Figure 6.2: Schematic of the control architecture of the experimental FMC



6.3 Computer-Based Control

The objective of the interactive control system proposed in this work is to transform the production requirements of a given automated manufacturing system into a set of control actions. These actions can then safely drive the system to fulfill these

requirements, while maintaining an optimized behavior. In accordance, the highest level of the proposed control hierarchy constitutes a *scheduler* that can transform the given production requirements into a deadlock-free and optimized production schedule. This scheduler is also capable of reacting to a wide range of system disruptions by modifying the deadlock-free schedule to accommodate these disruptions in real time. To this end, the MIP models, the operations insertion algorithm (OI), along with the reactive deadlock-free scheduling tool (GDRS), constitute the building blocks of the scheduler level in the control hierarchy.

6.3.1 Associating Control Actions and Conditions to the ASMG

In order to utilize a production schedule in supervising and driving an automated manufacturing system, it must be first transformed into a discrete-event supervisor, capable of driving the system in a closed loop. In Chapter 5, it has been shown how a deadlock-free schedule can be transformed into an ASMG. This ASMG captures all the required production requirements, and the safe optimized behavior that can drive the system to execute these requirements. However, in order for the ASMG to interact with the FMC in real time to realize the lower level control (*supervisor*) in the control hierarchy, its transitions must be associated with the I/O signals transmitted to/from the FMC. In other words, the execution of the FMC has to be associated with that of the ASMG, which is defined by the firing sequences of its transitions, in order to realize the required behavior. The simplest approach to reach this goal is to associate some of the transitions of the ASMG with control *actions* that can initiate events in the FMC, and

others with *conditions* in the FMC that have to be realized in order for these transitions to fire (recall automation PNs from Chapter 2). In accordance with the previously defined control scheme (Figure 6.2), the types of control action and condition (feedback) signals used in the experimental FMC are shown in Table 6.3.

Table 6.3: Control action signals and conditions

	Description	Symbol	Assoc. Transition
Action	Transport a job from P/D *j* to P/D *l* (44 signals)	$\alpha_{(j\text{-}l)}$	$t_{j\text{-}l}$
Condition	Robot available at home position	β_1	$t_{j\text{-}l}$
	Resource *j* (machine, central buffer) is acquired	$\beta_{2(j)}$	t_{Rj}

Recall that the presence of a token in an input scheduling place to a transition indicates that the corresponding machine has already been released by the previous job in its processing sequence. Hence, the availability of a machine *l* for acquisition in order for a transition $t_{j\text{-}l}$ to fire can be solely determined by the presence of a token in the input scheduling place to this transition. On the other hand, the availability of the robot for acquisition cannot be exclusively determined by the presence of a token in the robot place. This is because, a token in the robot place only indicates that the robot has been released from its previous handling operation, but does not indicate if the robot has returned to its home position or not. Accordingly, a feedback condition β_1 indicating the presence of the robot at its home position must be validated before firing a transition that re-acquires the robot.

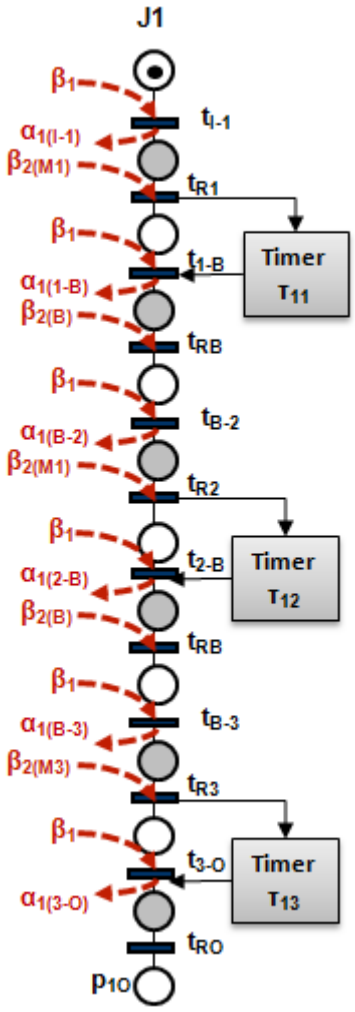
Because the machines present in the experimental FMC are virtual ones, where no actual processing takes place, a feedback signal from a machine indicating the completion of processing of a job cannot be attained. In order to overcome this concern in the experiments, each transition $t_{j\text{-}l}$ that models the completion of a processing operation of a job i on a machine j in the ASMG is associated with a *timer* τ_{ij}. The time associated with each of these transitions is equivalent to the processing time of job i on machine j. The timer τ_{ij} is initiated in the ASMG when the feedback signal $\beta_{2(j)}$ associated with the previous transition t_{Rj} in job i's PPN is received from the cell, indicating that job i has been delivered to machine j. When τ_{ij} expires, transition $t_{j\text{-}l}$ can then be enabled to fire.

To illustrate, consider the PPN of J_1 in the ASMG of problem '4J X 3M' shown in Figure 5.20. The transitions of this PPN can be associated with the action and condition signals, and the timers as shown in Figure 6.3. Transitions of the type $t_{I\text{-}l}$, $t_{j\text{-}l}$, or $t_{B\text{-}l}$ model the acquisition of the robot to transfer a job to a machine l. In Figure 6.3, transition $t_{I\text{-}1}$ (and similarly transitions $t_{B\text{-}2}$ and $t_{B\text{-}3}$) models the acquisition of the robot to transfer J_1 from the input buffer I to machine M_1 and the acquisition of M_1 as well. Thus, firing $t_{I\text{-}1}$ is conditioned by the feedback signal β_1. In order to execute the handling operation of J_1 between the input buffer and M_1, an action signal must be sent to the robot arm to initiate this operation. Accordingly, the action signal $\alpha_{1(I\text{-}1)}$ is associated with $t_{I\text{-}1}$.

Transitions of the type t_{Rj} model the delivery and the initiation of processing of a job on some machine j. Accordingly, transition t_{R1} (and similarly transitions t_{R2} and t_{R3}) in Figure 6.3 cannot fire before receiving the signal $\beta_{2(1)}$ that indicates that a job has been

delivered to M_1. Firing this transition, also initiates timer τ_{11} to simulate the processing of J_1 on M_1.

Figure 6.3: Association of transitions with action and condition signals and timers



Transitions of the type $t_{j\text{-}I}$, $t_{j\text{-}B}$, or $t_{j\text{-}O}$ model the completion of processing of a job on some machine j, the release of machine j, and the acquisition of the robot to deliver the job to machine l, the central buffer, or the output buffer, respectively. Firing transition $t_{1\text{-}B}$ (and similarly transitions $t_{2\text{-}B}$ and $t_{3\text{-}O}$) is then associated with the expiry of timer τ_{11}, the

β_1 condition indicating the availability of the robot, and the $\alpha_{1(1\text{-}B)}$ action signal for the robot to transfer J_1 from M_1 to the central buffer.

Finally, transitions t_{RB} model the delivery of a job to the central buffer. This type of transitions are associated with only one feedback condition $\beta_{2(B)}$, which indicates that the job has been placed in the central buffer. It can be noticed from Figure 6.3 that transition t_{RO} is not associated with any feedback condition. This is because in the current setting, it is assumed that the output buffer is the final stage for the job in the system, after which no further handling or processing operations will be performed. However, in the case where the FMC is part of a bigger manufacturing system, feedback conditions could be associated with such transitions to regulate the possible operation of other material handling devices that transfer the jobs between manufacturing cells.

6.3.2 Executing the ASMG

Executing a PN means firing its enabled transitions. In addition, the structure and the behavior of a PN can be defined by means of the incidence matrix (*IM*) and the state equation of the net, respectively (Appendix A). The state equation defines the next marking of the net based on the current firing vector *F* that lists the enabled transitions of the net. Hence, to execute the ASMG of a product mix and consequently the processing of this product mix in the manufacturing cell, first the *IM* is defined according to the structure of the net. Then, based on the marking *M* of the net and the feedback signals from the cell, *F* can be determined.

In this work, the *control algorithm* was developed in MATLAB using the MATLAB Data Acquisition Toolbox. In this algorithm, each of the control-action-initiating transitions is associated with the corresponding vector of the robot control input combinations (Table 6.1). When the corresponding element of any of these transitions in *F* takes a value of 1, and the net fires, the corresponding input combination is sent to the I/O module to execute the correct handling operation. The state equation of the net then updates the marking of the net. The control algorithm repeats this process until all the jobs in the cell are completed. It should be noted that, when more than one control-action-initiating transition is enabled in the net, only one of these transitions is chosen *randomly* to fire.

To illustrate, re-consider the PPN shown in Figure 6.3. In this PPN, transition t_{I-1} models the transportation of J_1 from the input buffer to M_1. According to the current marking of the net, t_{I-1} is the only *net-enabled* transition in the net. However, in order for this transition to fire, it must be also enabled by the cell. In other words, the value of β_1 must be equal to one, indicating that the robot arm is at its home position and ready to perform a transportation operation. Assuming that $\beta_1 = 1$, then the firing vector of this PPN will be $[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$. Firing t_{I-1} is associated with the control input combination number 1: [1 0 0 0 0 0]. Hence, when the net fires, the command function *putvalue* (Table 6.2) will set the values of output lines (1-6) of the I/O module according to the selected combination. The values of these lines will then be input to the robot arm controller to call sub-program 1 and execute the required handling operation.

6.4 Implementation and Experimentation

In order to validate the correctness of all the levels of the proposed control hierarchy, and to test whether there were any software or hardware problems in the setting of the experimental FMC, nine problems are randomly generated and executed in the FMC. The nine problems all feature the processing of five different job types on the four virtual machines. Three of the problems are generated such that the jobs have short processing times on the machines, three have medium processing times, and three have long processing times. The processing times for the jobs on the machines are generated from uniform distributions of (1,5), (5,10), and (10,15) seconds for the short, medium, and long processing time problems, respectively. These values were selected in accordance with the actual times required by the robot arm to perform each material handling task within the experimental FMC, which are shown in Table 6.4.

Table 6.4: Handling times of robot arm (seconds)

FROM / TO	M_1	M_2	M_3	M_4	CB	OB
I_A	5.05	4.59	4.95	5.08	N/A	N/A
I_B	4.31	4.52	4.73	4.91	N/A	N/A
I_C	4.08	4.28	4.53	4.66	N/A	N/A
I_D	3.98	4.20	4.39	4.66	N/A	N/A
I_E	4.39	4.55	4.59	4.83	N/A	N/A
M_1	0.00	4.67	5.11	5.13	5.03	5.91
M_2	4.86	0.00	4.95	5.11	4.95	6.06
M_3	5.13	4.97	0.00	5.17	5.09	6.50
M_4	5.30	5.20	5.02	0.00	5.13	6.39
CB	5.44	5.47	5.55	5.45	0.00	N/A

The steps required to execute each of the nine problems (product mixes) in the FMC are set in accordance with the proposed control hierarchy as follows:

1. Solve the deadlock-free scheduling problem for the given product mix to obtain the best schedule of the five jobs on the four machines.
2. Transform the obtained schedule into the corresponding SMG.
3. Realize the controller ASMG from the SMG.
4. Define the control actions and conditions associated with each transition in the ASMG.
5. Execute the ASMG using the control algorithm.

6.4.1 Experimental Results

Using the proposed insertion algorithm (OI) to perform step 1 of the execution steps, the time required to reach step 5 for each of the nine problems was less than one second. Furthermore, the nine product mixes were executed in the experimental FMC, and were all completed flawlessly.

To assess the performance of the control architecture and the experimental setting, three experiments are performed using the nine problems. The objective of the first is to examine the consistency of the control algorithm and setting in repeating the execution of the same product mix more than once. This is achieved by collecting the actual times required to execute one product mix from each of the three groups of problems for three consecutive times. The objective of the second experiment is to observe the robot arm

utilization and the deviation between the theoretical makespan obtained using OI and that realized upon execution for each of the nine problems. In the third experiment, the same metrics are measured when the material handling operations are considered in the schedule building process using OI and the transportation operations insertion algorithm. The results for experiments 1, 2 and 3 are shown in Tables 6.5, 6.6, and 6.7, respectively.

Table 6.5: Results of Experiment 1

	Fast Process. Time	Med. Process. Time	Slow Process. Time
	Actual Makespan (sec)		
Trial 1	166.60	165.14	213.12
Trial 2	166.95	165.03	213.14
Trial 3	166.45	165.09	213.11

Table 6.6: Results of Experiment 2

	Fast Process. Time			Med. Process. Time			Slow Process. Time		
	RU	AMS	TMS	RU	AMS	TMS	RU	AMS	TMS
Problem 1	87.1	166.60	26	85.2	165.14	62	68.4	213.12	113
Problem 2	85.0	178.50	21	80.5	175.00	64	65.8	221.76	109
Problem 3	85.3	164.90	22	79.9	177.23	59	74.5	195.26	101

*** RU: Robot Utilization (%), AMS: Actual Makespan (sec), TMS: Theoretical Makespan (sec)**

Table 6.7: Results of Experiment 3

	Fast Process. Time			Med. Process. Time			Slow Process. Time		
	RU	AMS	TMS	RU	AMS	TMS	RU	AMS	TMS
Problem 1	88.3	152.4	136.8	83.4	161.8	144.0	66.0	212.3	202.33
Problem 2	86.3	150.6	129.9	83.8	173.8	153.3	74.1	197.1	194.5
Problem 3	89.1	158.1	139.4	80.0	177.15	163.0	72.4	193.4	186.0

*** RU: Robot Utilization (%), AMS: Actual Makespan (sec), TMS: Theoretical Makespan (sec)**

From Table 6.5, the maximum deviation in actual makespan between the three trials (repetitions) for the three tested problems is 0.3% (fast processing time problem). This shows that the utilized control algorithm and setting (hardware) can adequately realize and preserve the control scheme embedded in the ASMG.

Table 6.6 shows that the deviation between the actual makespans and the expected ones is vast, and is more significant in the fast and medium problems. This is because, in these trials, the material handling times were not considered in the schedule building process. In addition, as can be noticed from Table 6.4, these times are larger than and almost equivalent to the processing times for the fast and medium problems, respectively. This is also why, although the theoretical makespans are considerably larger in medium problems than in the fast ones, the actual makespans for the fast and medium problems are very close in value.

Table 6.7 shows that the deviation between the actual makespans and the theoretical ones is drastically reduced when the material handling operations are considered in the schedule building process. Nevertheless, some deviations still remain. These deviations can be mainly attributed to the relatively slow scanning speed of the utilized robot arm controller; it sometimes takes more than a second to call the correct sub-program. A second cause for these deviations can be the random selection of a transition to fire during execution when more than one is enabled. This table also shows that considering the transportation operations in the scheduling process can reduce the actual makespan realized on the shop-floor. This can be deduced by comparing the actual makespan values in Tables 6.6 and 6.7, especially for the fast processing time problems.

Finally, Tables 6.6 and 6.7 show that the robot arm is better utilized in the fast and medium processing time product mixes than in the slow ones. This is expected since in the slow ones, the robot arm remains idle for considerable amounts of time while waiting for the processing of the jobs on the machines (simulated in the ASMG) to be completed.

6.5 Conclusions

In this chapter, the proposed control hierarchy was validated and implemented in an experimental FMC. The FMC constituted a PC that represented the upper two levels of the control hierarchy (scheduler and supervisor), an I/O data acquisition module, a relay box, a robot arm, four virtual machines, five input buffers for five job types, a central buffer with a unit capacity, and an output buffer. It was shown how the robot arm was efficiently controlled using a combinatorial control scheme that utilized six input lines to its controller to execute 44 different material handling operations. The utilization of the I/O module and MATLAB's Data Acquisition toolbox to relay the control signals between the PC and the other cell components was also illustrated. Furthermore, the types of control action and feedback signals that can be exchanged in such a manufacturing environment were also discussed. In addition, the association of each type of these signals with the transitions of the controller ASMG was shown.

The proposed control scheme was implemented in the FMC to execute nine product mixes. All the mixes were successfully completed in the cell with no faults or deadlocks. The time required to realize an executable controller for each of the given mixes was less

than one second. This demonstrates the robustness and agility of the proposed control architecture in dealing with today's dynamic production environment. Indeed, by using an ordinary PC and an off-the-shelf I/O digital module, efficient control algorithms for complex job shop automated systems can be obtained instantly.

The experimental results showed that the utilized simple control setting can adequately realize the generated control algorithms and preserve their consistency. They also showed that considering the transportation operations in the schedule building process can more accurately estimate and can reduce the actual makespans realized on the shop-floor. Finally, the under-utilization of the robot arm in slow processing time mixes can be efficiently exploited to increase the life expectancy of the arm components by reducing the movement speed of the arm.

CHAPTER 7:

Conclusions and Recommendations

The justification of fully automated systems in industry necessitates the availability of efficient tools that can realize the best production outcome from such systems. The investments required to install highly versatile CNC machines, flexible robot manipulators or other autonomous material handling devices like automated guided vehicles (AGVs), and automated storage and retrieval systems (ASRS), are high but are needed to compete in the industrial market. Failing to utilize the foremost capabilities of such resources can often result in denouncing their vital role in industry advancement.

Flexibility in production essentially refers to the capability of coping with and reacting to the ever-changing customer needs and the strong competition in today's global market. If there is a single common feature between versatile CNCs, robotic manipulators, AGVs, and ASRSs, it would indeed be flexibility; flexibility in performing various manufacturing processes on a single resource, flexibility in processing and delivering any part from/to any resource in the system at any time, flexibility of providing the required part type or machine tool at any time, and many more types of other flexibilities. Competition in today's markets essentially requires these types of flexibilities. Nevertheless, wide implementation of such flexibilities in industry has been impeded by

undermined performance and autonomous control complexities, and has yet to be realized.

7.1 Research Contributions

This research has proposed an efficient hierarchical scheduling and control architecture for flexible automated manufacturing systems. The inputs to the proposed architecture are simply the available resources in the system, the production routes of the jobs to be produced, and any sudden internal or external disruptions to the system that may occur during production. The final output is a control algorithm, capable of driving the system to autonomously produce the required products in a deadlock-free manner, according to the best production schedule that optimizes the performance measures of the system. The control algorithm can further be readily updated in real time to accommodate any changes in the product mix or the production conditions, while preserving the optimized performance and maintaining the stability of the system. The hierarchical architecture developed in this thesis is an integration of a number of tools as outlined below:

- Four MIP models; IBS, IB1, IBA, and CBA. These four models can provide the optimal deadlock-free production schedules for flexible job shop systems, considering four different buffer configurations that may exist in a typical manufacturing environment. The models can be used to optimize any completion time-related objective criteria; these include minimizing the mean

flow time (section 3.2), minimizing the makespan, maximizing machine utilization, and minimizing the mean tardiness in the system.

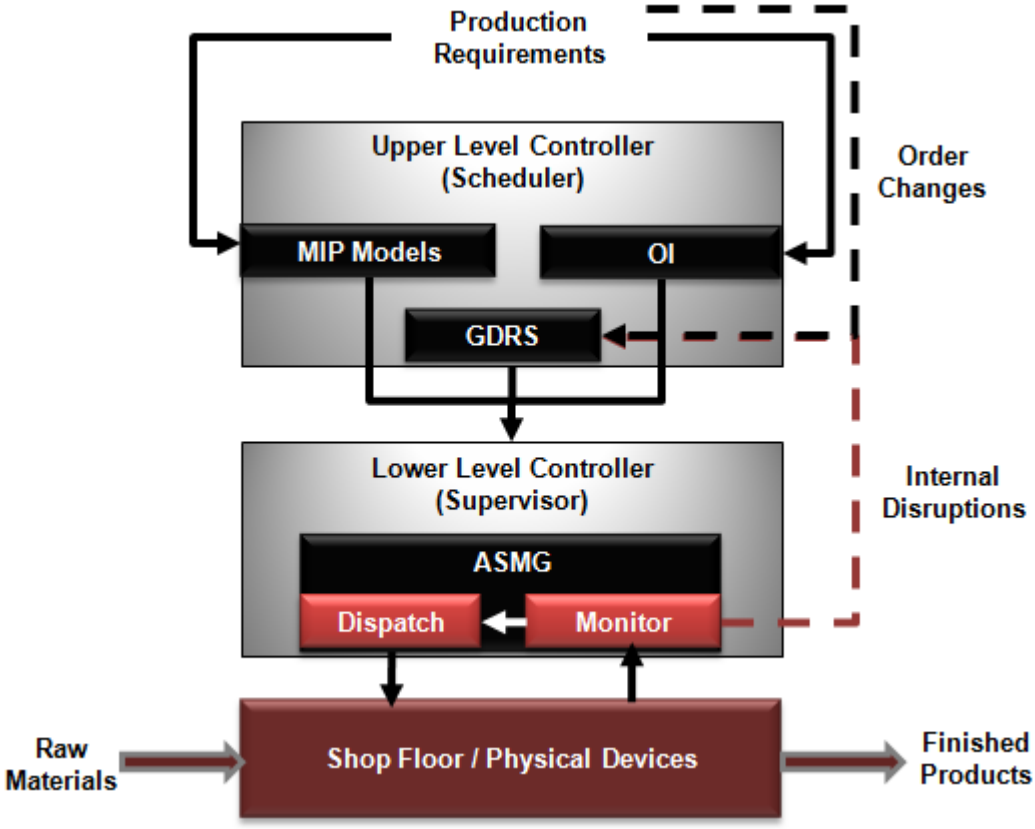
- An operations insertion algorithm (OI) that can provide optimal or near optimal deadlock-free production schedules with regard to the same objective criteria indicated above, in near real time. The algorithm utilizes the rank matrices of the schedules to detect and resolve any deadlock situations using the available buffer space (if any).
- A generic deadlock-free reactive scheduling tool (GDRS) that can react to five main types of disruptions that can occur in a manufacturing system, namely; arrival of new jobs, machine breakdowns, processing time variations, urgency of existing jobs, and order cancellations. The main building block of GDRS is OI, and is proved to have the capability of producing reactive schedules that feature optimized performance and preserved system stability.
- A formal procedure to transform a production schedule into an augmented scheduling marked graph (ASMG). This ASMG can translate a production schedule into a set of discrete event commands to realize the optimized and correct performance attained by the schedule on the shop floor.

These four components were integrated to achieve the proposed scheduling and control architecture as shown in Figure 7.1.

Implementation experiments were conducted in an experimental manufacturing cell to validate the performance and correctness of the proposed architecture. The results showed that, using an ordinary PC, an off-the-shelf digital I/O module, and a set of

relays, the proposed architecture can be implemented and readily updated to schedule and autonomously control the production of a number of job types, with complex routes, in a typical flexible manufacturing environment.

Figure 7.1: Integration of proposed tools in the proposed architecture



7.2 Conclusions

In the previous chapters, the proposed tools were introduced in detail and evaluated for performance. In Chapter 6, it was shown how these tools can be integrated to realize efficient and correct production behavior on the shop floor. The following points summarize the main achievements and results attained in this research:

1. The proposed MIP models provided novel constraints that could represent and utilize the available buffer space in the job shop environments considered, and this was previously unachievable. The performance of these models was compared to previously proposed models in literature in terms of both solution quality and time. Results showed that the proposed models can obtain better schedules due to efficient utilization of any available buffer space.
2. The operations insertion algorithm (OI) can handle a wide variety of parameters in the deadlock-free scheduling problem, while providing optimal or near optimal schedules, almost in real time. The performance of OI was compared to the performance of a number of approaches proposed in literature, and the results showed that in most of the cases, OI either obtained the same or better results than those achieved by other approaches in a timely efficient manner.
3. The proposed transportation operations insertion algorithm (TOI) can either insert transportation operations after obtaining the optimal schedules, or augment these operations in the schedule building process when using OI. It preserved the deadlock-freeness of the resulting schedules while providing a material handling schedule that conforms with the production schedule.
4. Because of solution time considerations, the MIP models can be used to solve small and fairly medium-sized problems (Section 3.6.1) to optimality, whereas OI can be used to solve larger problems. Nevertheless, the MIP models can determine the optimal buffer configuration that can tolerate the best achievable

production practice for medium-sized systems during the system set-up phase, when solution time is not a considerable issue.

5. To the best of the author's knowledge, GDRS is the first reactive scheduling tool that can react to a wide variety of production disruptions while preventing the deadlock inevitability in automated job shops. The performance of GDRS was compared to an approach proposed earlier that can be utilized to react to the types of disruptions considered (mAOR), and to the total rescheduling approach (TR) that reschedules all the operations in the system to preserve efficiency with no consideration for system stability. The results demonstrated the overall efficiency of GDRS over mAOR, and showed that the performance of GDRS is comparable to that of TR in most disruptions, but with a considerable improvement in system stability.
6. The idea of transforming a schedule into a supervisor has been rarely discussed in previous literature, with no actual application. The proposed approach that transforms deadlock-free schedules into ASMGs is novel and straightforward. Using this approach, a readily implementable controller can be obtained for considerably complex systems in a few simple steps, not only in a fraction of the time required by previous approaches, especially the ones based on automata, but also in a manner that realizes the best performance of the system.
7. The utilized implementation scheme is simple, yet effective. The set-up required a PC and a few off-the-shelf components to realize a complex control practice for a complex type of manufacturing systems.

The demonstrated effectiveness of the proposed control architecture, along with the simplicity in the required shop floor implementation scheme, can advocate and promote the wider utilization of automated flexible manufacturing systems in industry.

7.3 Recommendations

Since the scheduler level of the control architecture is actually the one that determines the final system behavior on the shop floor, improving the performance of the components of this level (MIP models and OI) can enhance the performance of the whole architecture. As for the MIP models, they are capable of attaining the optimal schedules under the assumptions considered for the tested problem sizes. However, the solution time required to solve these models remain an obstacle when considering medium and large systems. To avert this problem, an approximation heuristic can be developed to solve these models to obtain near optimal schedules in a timely efficient manner. Such heuristics have been commonly used in previous literature to approximate complex MIP models, mainly by developing the Lagrangian relaxation of the models.

As for OI, the order of insertion of the jobs in the schedule building process, can highly affect the final outcome. In OI, this order was mainly based on the processing routes of the jobs, with the objective of deferring any conflicts in these routes to the final stages of the schedule building process. A more global approach that can base this order on both the processing routes and also the potential performance of the final schedule can be developed to tackle this issue. This approach can feature a meta-heuristic algorithm

like GA, in which the deadlock detection and resolution techniques can be embedded. Hence, instead of the current greedy nature of OI that schedules the jobs one after the other, a more global approach can be utilized to obtain schedules with better values of the performance measures.

7.3.1 Plant-Wide Control

The outcomes of this research have been mainly proposed, studied, discussed, and implemented in a cellular level that features a single material handler. While these outcomes can be beneficially utilized in systems featuring a single production cell, larger systems that constitute multiple cells require a global plant-wide level of control. An important research direction would then be to expand the current work to acquire global controllers capable of driving a multi-cell manufacturing system in an autonomous, optimized manner. This controller would ensure feasible and conflict-free interactions within and between the cells, and concurrently guarantee the optimized performance of the whole system. This can be achieved by firstly developing an algorithm to concurrently optimize the cell formation and the deadlock-free scheduling problems in the system design phase. An algorithm to schedule the utilization of inter-cell common resources, like material handling devices, in a manner that ensures deadlock-free operation, would then be developed. The obtained schedules can be eventually augmented into a global system controller.

References

[1] Sun, R.-L., Li, H.-X. and Xiong, Y., 2006. Performance-oriented integrated control of production scheduling. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no. 4, pp.554-562.

[2] Lin, J. T. and Lee, C.-C., 1997. A Petri net-based integrated control and scheduling scheme for flexible manufacturing system. *Computer Integrated Manufacturing Systems*, vol. 10, no. 2, pp. 109-122.

[3] Aytug, H., Lawley, M. A., McKay, K., Mohan, S., and Uzsoy, R., 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, vol. 161, pp. 86–110.

[4] Li, L. and Jiang, Z., 2006. Formal design and analysis of a hybrid supervisory control structure for virtual production systems. *International Journal of production Research*, vol. 44, no.13, pp. 2479-2497.

[5] Mohan, S., Yalcin, A., and Khator, S., 2004. Controller design and performance evaluation for deadlock avoidance in automated flexible manufacturing cells. *Robotics and Computer Integrated Manufacturing*, vol. 20, no. 6, pp. 541-551.

[6] Ghaffari, A., Rezg, N. and Xie, X., 2003. Feedback control logic for forbidden-state problems of marked graphs: application to a real manufacturing system. *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 18-29.

[7] Yalcin, A., 2004. Supervisory control of automated manufacturing cells with resource failures. *Robotics and Computer-Integrated Manufacturing*, vol. 20, no.2, pp. 111-119.

[8] Ramadge, P. J. and Wonham, W. M., 1987. Supervisory control of a class of discrete event processes. *Siam J. Control and Optimization*, vol. 25, no. 1, pp. 206-230.

[9] Wonham, W. M and Ramadge, P. J., 1987. On the supremal controllable sublanguage of a given language. *Siam J. Control and Optimization*, vol. 25, no. 3, pp. 637-659.

[10] Ramadge, P. J. and Wonham, W. M., 1989. The control of discrete event systems. *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81-89.

[11] Alpan, G. and Jafari, M. A., 2002. Synthesis of a closed-loop combined plant and controller module. *IEEE Transaction on Systems, Man, and Cybernetics*, vol. 32, no. 2, pp. 163-175.

[12] Brandin, B. A., 1996. The real-time supervisory control of an experimental manufacturing cell. *IEEE Transactions on Robotics and Automation*, vol. 12, no. 1, pp. 1-14.

[13] Ramirez-Serrano, A. and Benhabib, B., 2003. Supervisory control of functionally-expandable manufacturing systems. *The International Journal of Flexible Manufacturing Systems,* vol. 15, no.3, pp. 241-271.

[14] Nourelfath, M. and Niel, E., 2004. Modular supervisory control of an experimental automated manufacturing system. *Control Engineering Practice,* vol. 12, no.2, pp. 205-216.

[15] Uzam, M., Jones, H. A. and Yűcel, I., 2000. Using a Petri-net-based approach for the real-time supervisory control of an experimental manufacturing system.

International Journal of Advanced Manufacturing Technology, vol. 16, no. 7, pp. 498-515.

[16] Uzam, M. and Wonham W. M., 2006. A hybrid approach to supervisory control of discrete event systems coupling RW supervisors to Petri nets. *International Journal of Advanced Manufacturing Technology*, vol. 28, no. (7-8), pp. 747-760.

[17] Brandin, B. A. and Wonham, W. M., 1993. Modular supervisory control of timed discrete-event systems. *Proceedings of the 32nd Conference on Decision and Control*, vol. 3, pp. 2230-2235.

[18] Wonham, W. M., 2005. Supervisory control of discrete event systems. © Copyright by W. M. Wonham, 1997-2005.

[19] Barrett, G. and Lafortune, S., 2000. Decentralized supervisory control with communicating controllers. *IEEE Transactions on Automatic Control,* vol. 45, no. 9, pp. 1620-1638.

[20] Yoo, T.-S. and Lafortune, S., 2004. Decentralized supervisory control with conditional decisions: supervisor existence. *IEEE Transactions on Automatic Control,* vol. 49, no. 11, pp. 1886-1904.

[21] Yoo, T.-S. and Lafortune, S., 2005. Decentralized supervisory control with conditional decisions: supervisor realization. *IEEE Transactions on Automatic Control,* vol. 50, no. 8, pp. 1205-1211.

[22] Leduce, R. J., Lawford, M. and Dai, P., 2006. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Transactions on Control Systems Technology*, vol. 14, no.4, pp. 654-668.

[23] Brandin, B. A., Wonham, W. M. and Benhabib, B., 1992. Manufacturing cell supervisory control – A timed discrete event system approach. *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, pp. 931-936.

[24] Brandin, B. A. and Wonham, W. M., 1994. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control,* vol. 39, no. 2, pp. 329-342.

[25] Golmakani, H. R., Mills, J. K. and Benhabib, B., 2003. Deadlock-free scheduling of flexible manufacturing workcells using automata theory. *Proceedings of the IEEE International* Conference on Robotics and Automation, vol. 1, pp. 169-174.

[26] Golmakani, H. R., Mills, J. K. and Benhabib, B., 2006. Deadlock-free scheduling and control of flexible manufacturing cells using automata theory. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no.2, pp. 327-337.

[27] Golmakani, H. R., Mills, J. K. and Benhabib, B., 2006. On-line scheduling and control of flexible manufacturing cells using automata theory. *International Journal of Computer Integrated Manufacturing*, vol. 19, no. 2, pp. 178-193.

[28] Fanti, M. P. and Zhou, M., 2004. Deadlock control methods in automated manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 34, no.1, pp. 5-32.

[29] Viswanadham, N., Narahari, Y. and Johnson, T. L., 1990. Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models. *IEEE Transactions on Robotics and Automation,* vol.6, no. 6, pp. 713-723.

[30] Banaszak, Z. A. and Krogh, B. H., 1990. Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Transactions on Robotics and Automation,* vol. 6, no. 6, pp. 724-734.

[31] Hsieh, F.-S. and Chang, S.-C., 1994. Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation,* vol.10, no. 2, pp. 196-209.

[32] Zhou, M. C. and DiCesare, F., 1991. Parallel and sequential mutual exclusions for Petri net modeling of manufacturing systems with shared resources. *IEEE Transactions on Robotics and Automation,* vol.7, no. 4, pp. 515-527.

[33] Zhou, M. C., DiCesare, F. and Derochers, A. A., 1992. A hybrid methodology for synthesis of Petri nets for manufacturing systems. *IEEE Transactions on Robotics and Automation,* vol.8, no. 3, pp. 350-361.

[34] Ezpeleta, J., Colom, J. M. and Martinez, J., 1995. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation,* vol. 11, no. 2, pp. 173-184.

[35] Chu, F. and Xie, X.-L., 1997. Deadlock analysis of Petri nets using siphons and mathematical programming. *IEEE Transactions on Robotics and Automation,* vol. 13, no. 6, pp. 793-804.

[36] Ben Abdallah, B., ElMaraghy, H. A. and ElMekkawy, T., 1997. A logic programming approach for finding minimal siphons in S^3PR nets applied to manufacturing systems. *Proceedings of the IEEE Conference on Systems, Man and Cybernetics,* vol. 2, pp. 1710-1715.

[37] Ben Abdallah, I. and ElMaraghy, H. A., 1998. Deadlock prevention and avoidance in FMS: A Petri net based approach. *International Journal of Advanced Manufacturing Technology*, vol. 14, no. 10, pp. 704-715.

[38] Park, J. and Reveliotis, S. A., 2001. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Transactions on Automatic Control*, vol. 46, no. 10, pp. 1572-1583.

[39] Tricas, F. and Ezpeleta, J., 2006. Computing minimal siphons in Petri net models of resource allocation systems: A parallel solution. *IEEE Transactions on Systems, Man and Cybernetics,* vol. 36, no. 3, pp. 532-539.

[40] Li, Z. and Wei, N., 2007. Deadlock control of flexible manufacturing systems via invariant-controlled elementary siphons of Petri nets. *International Journal of Advanced Manufacturing Systems,* vol. 33, no. 1-2, pp. 24-35.

[41] Li, Z. W. and Zhou, M. C., 2004. Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics,* vol. 34, no. 1, pp. 38-51.

[42] Li, Z. W., Hu, H. and Zhou M. C., 2004. An algorithm for an optimal set of elementary siphons in Petri nets for deadlock control. Proceedings of the *IEEE Conference on Systems, Man and Cybernetics,* vol. 5, pp. 4849-4854.

[43] Zhou, M. C., DiCesare, F. and Rudolph, D. L., 1992. Design and implementation of a Petri net based supervisor for a flexible manufacturing system. *Automation*, vol. 28, no. 6, pp. 1199-1208.

[44] Lee, D. L. and DiCesare, F., 1994. Scheduling flexible manufacturing systems using Petri nets and heuristic search. *IEEE Transactions on Robotics and Automation,* vol.10, no. 2, pp. 123-132.

[45] Xiong, H. H., Zhou, M. C. and Caudil, R. J., 1996. A hybrid heuristic search algorithm for scheduling flexible manufacturing systems. *Proceedings of the IEEE International Conference on Robotics and* Automation, vol. 3, pp. 2793-2979.

[46] Xiong, H. H. and Zhou, M. C., 1997. Deadlock-free scheduling of an automated manufacturing system based on Petri nets. *Proceedings of the IEEE International Conference on Robotics and Automation,* vol. 2, pp. 945-950.

[47] Jeng, M. D., Chiou, W. D. and Wen, Y. L., 1998. Deadlock-free scheduling of flexible manufacturing systems based on heuristic search and Petri net structures. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pp. 26-31.

[48] Ben Abdallah, I., ElMaraghy, H. A. and ElMekkawy, T., 1998. An efficient search algorithm for deadlock-free scheduling in FMS using Petri nets. *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1793-1798.

[49] Ben Abdallah, I., ElMaraghy, H. A. and ElMekkawy, T., 2002. Deadlock-free scheduling in flexible manufacturing systems using Petri nets. *International Journal of Production Research*, vol. 40, no. 12, pp. 2733-2756.

[50] Tamayo, A. J., Contreras, D. G. and Trevino, A. R., 1998. Petri net based control for the dynamic scheduling of a flexible manufacturing cell. *Proceedings of the*

IEEE International Conference on Systems, Man, and Cybernetics, vol. 1, pp. 553-557.

[51] Huang, Z. and Wu, Z., 2004. Deadlock-free scheduling method for automated manufacturing systems using genetic algorithm and Petri nets. *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, pp. 566-571.

[52] Huang, Z. and Wu, Z., 2004. Deadlock-free scheduling in automated manufacturing systems with multiple resource requests. *IEICE Transactions on Fundamentals*, vol. E87-A, no. 11, pp. 2844-2851.

[53] Shi, X. and Wu, Z., 2005. Deadlock-free scheduling methods for FMSs using beam search. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, pp. 1188-1193.

[54] Damasceno, B. C. and Xie, X., 1998. Scheduling and deadlock avoidance of a flexible manufacturing system. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 564-569.

[55] Damasceno, B. C. and Xie, X., 1999. Petri nets and deadlock-free scheduling of multiple resource operations. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 878-883.

[56] Liljenvall, T., 1999. Scheduling for production systems with limited buffers. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 6, pp. 469-474.

[57] Huang, Z. and Wu, Z., 2004. Deadlock-free scheduling method for automated manufacturing systems with limited central buffers. *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, pp. 560 – 565.

[58] Mati, Y., Rezg, N. and Xie, X., 2001. A taboo search approach for deadlock-free scheduling of automated manufacturing systems. *Journal of Intelligent Manufacturing*, vol. 12, pp. 535- 552.

[59] Mati, Y., Rezg, N. and Xie, X., 2001. Geometric approach and Taboo search for scheduling flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 805-818.

[60] Baker, K. B., 1974. Introduction to scheduling and sequencing. *Wiley*, New York.

[61] Pan, J. C.-H. and Chen, J.-S., 2005. Mixed binary integer programming formulations for the reentrant job shop scheduling problem. *Computers and Operations Research*, vol. 32, no. 5, pp. 1197-1212.

[62] Sawik, T., 2000. Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. *Mathematical and Computer Modeling*, vol. 31, no. 13, pp. 39-52.

[63] Gomes, M.C., Barbosa-Povoa, A.P. and Novais, A.Q., 2005. Optimal scheduling for flexible job shop operation. *International Journal of Production Research*, vol. 43, no. 11, pp. 2323-2353.

[64] Liao, C.-J. and You, C.-T., 1992. An improved formulation for the job-shop scheduling problem. *Journal of Operational Research Society*, vol. 43, no. 11, pp. 1047-1054.

[65] Kim, K.-H. and Egbelu P.J., 1998. A mathematical model for job shop scheduling with multiple process plan consideration per job. *Production Planning and Control*, vol. 9, no. 3, pp. 250–259.

[66] Kim, K.-H. and Egbelu, P.J., 1999. Scheduling in a production environment with multiple process plans per job. *International Journal of Production Research*, vol. 37, no. 12, pp. 2725-2753.

[67] Tamaki, H., Ono, T., Murao, H. and Kitamura, S., 2001. Modeling and genetic solution of a class of flexible job shop scheduling problems. *IEEE Symposium on Emerging Technologies and Factory Automation*, vol. 2, pp. 343-350.

[68] Low, C. and Wu, T.-H., 2001. Mathematical modeling and heuristic approaches to scheduling problems in an FMS environment. *International Journal of Production Research*, vol. 39, no. 4, pp. 689-708.

[69] Low, C., Wu, T.-H. and Hsu, C.-H., 2005. Mathematical modeling of multi-objective job shop scheduling with dependant setups and re-entrant operations. *International Journal of Advanced Manufacturing Technology*, vol. 27, no. (1-2), pp. 181-189.

[70] Ramaswamy, S. E. and Joshi, S. B., 1996. Deadlock-free schedules for automated manufacturing workstations. *IEEE Transactions on Robotics and Automation,* vol. 12, no.3, pp. 391-399.

[71] Artigues, C. and Roubellat, F., 2000. A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research*, vol. 127, no. 2, pp. 297-316.

[72] Artigues, C. and Roubellat, F., 2002. An efficient algorithm for operation insertion in a multi-resource job-shop schedule with sequence-dependant setup times. *Production Planning & Control*, vol. 13, no. 2, pp. 175-186.

[73] Ourari, S. and Bouzouia, B., 2003. An approach based on operation insertion for the one-machine real-time scheduling. *International Journal of Robotics and Automation*, vol. 18, no. 4, pp. 185 – 190.

[74] Kis, T. and Hertz, A., 2003. A lower bound for the job insertion problem. *Discrete Applied Mathematics*, vol. 128, no. (2-3), pp. 395-419.

[75] Sotskov, Y. N., Tautenhahn, T. and Werner, F., 1999. On the application of insertion techniques for job shop problems with setup times. *RAIRO Recherche Operationnelle*, vol. 33, no. 2, pp. 209 – 245.

[76] Werner, F. and Winkler, A., 1995. Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, vol. 58, no. 2, pp. 191 – 211.

[77] Gröflin, H. and Klinkert, A., 2007. Feasible insertions in job shop scheduling, short cycles and stable sets. *European Journal of Operational Research,* vol. 177, no. 2, pp. 763 – 785.

[78] Subramaniam, V., Raheja, A. S. and Reddy, K. R. B., 2005. Reactive repair tool for job shop schedules. *International Journal of Production Research*, vol. 43, no. 1, pp. 1-23.

[79] Chiang, T.-W. and Hau, H.-Y., 1996. Solving job insertion problem in job shop scheduling using iterative improvement. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 2, pp. 1525–1530.

[80] Hall, N. G. and Potts, C. N., 2004. Rescheduling for new orders. *Operations Research*, vol. 52, no.3, pp. 440-453.

[81] Branke, J. and Mattfeld, D. C., 2005. Anticipation and flexibility in dynamic scheduling. *International Journal of Production Research*, vol. 15, no. 15, pp. 3103-3129.

[82] Li, L. and Jiang, Z., 2007. Self-adaptive dynamic scheduling of virtual production systems. *International Journal of production Research*, vol. 45, no.9, pp. 1937-1951.

[83] Abumaizar, R. J. and Svestka, J. A., 1997. Rescheduling job shops under random disruptions. *International Journal of Production Research*, vol. 35, no. 7, pp. 2065-2082.

[84] ElMekkawy, T. Y. and ElMaraghy, H. A., 2003. Real-time scheduling with deadlock avoidance in flexible manufacturing systems. *International Journal of Advanced Manufacturing Technology*, vol. 22, no. (3-4), pp. 259-270.

[85] Jensen, M. T., 2003. Generating robust and flexible job shop schedules using Genetic algorithms. *IEEE Transactions on Evolutionary Computations*, vol. 7, no. 3, pp. 275-288.

[86] Mason, S. J., Jin, S., and Wessels, C. M., 2004. Rescheduling strategies for minimizing total weighted tardiness in complex job shops. *International Journal of Production Research*, vol. 42, no.3, pp. 613-628.

[87] Bollapragada, R. and Sadeh, N. M., 2004. An empirical study of policies to integrate reactive scheduling and control in just-in-time job shop environments. *International Journal of Production Research*, vol. 42, no. 4, pp. 693-718.

[88] Suwa, H. and Sandoh, H., 2007. Capability of cumulative delay based reactive scheduling for job shops with machine breakdowns. *Computers and Industrial Engineering*, vol. 53, no. 1, pp. 63-78.

[89] Sabuncuoglu, I. and Kizilisik, O. B., 2003. Reactive scheduling in a dynamic and stochastic FMS environment. *International Journal of Production Research*, vol. 41, no. 17, pp. 4211-4231.

[90] Subramaniam, V. and Raheja, A. S., 2003. mAOR: A heuristic-based reactive repair mechanism for job shop schedules. *International Journal of Advanced Manufacturing Technology*, vol. 22, no. (9-10), pp. 669-680.

[91] Jain, A. K. and ElMaraghy H., 1997. Production scheduling/rescheduling in flexible manufacturing. *International Journal of Production Research*, vol. 35, no. 1, pp. 281-309.

[92] Honghong, Y. and Zhiming, W., 2003. The application of Adaptive Genetic Algorithms in FMS dynamic rescheduling. *International Journal of Computer Integrated Manufacturing*, vol. 16, no. 6, pp. 382-397.

[93] Shi-jin, W., Li-feng, X., and Bing-hai, Z., 2007. Filtered-beam-search-based algorithm for dynamic rescheduling in FMS. *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 4, pp. 457-468.

[94] Raheja, S.A. and Subramaniam V., 2002. Reactive recovery of job shop schedules – a review. *International Journal of Advanced Manufacturing Technology*, vol. 19, no. 10, pp. 756-763.

[95] Unal, A.T., Uzsoy, R., and Kiran, A. S., 1997. Rescheduling on a single machine with part-type dependant setup times and deadlines. *Annals of Operations Research*, vol. 70, pp. 93–113.

[96] Ourari, S. and Bouzouia, B., 2003. An approach based on operation insertion for the one-machine real-time scheduling. *International Journal of Robotics and Automation*, vol. 18, no. 4, pp. 185-190.

[97] Liu, S. Q., Ong, H. L., and Ng, K. M., 2005. Metaheuristics for minimizing the makespan of the dynamic shop scheduling problem. *Advances in Engineering Software*, vol. 36, no. 3, pp. 199-205.

[98] Rovithakis, G. A., Perrakis, S. E., and Christodoulou, M. A., 2001. Application of a neural network scheduler on a real manufacturing system. *IEEE Transactions on Control Systems Technology*, vol. 9, no. 2, pp. 261-270.

[99] Chen, Y. L., Sun, T. H. and Fu, L. C., 1994. A Petri-net based hierarchical structure for dynamic scheduler of an FMS: rescheduling and deadlock avoidance. *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1998-2004.

[100] Ferrarini, L. and Poroddi, L., 2003. Modular design and implementation of a logic control structure for a batch process. *Computers and Chemical Engineering*, vol. 27, no. 7, pp. 983-996.

[101] Feldman, K., Colombo, A. W., Schnur, C. and Stőckel, T., 1999. Specification, design, and implementation of logic controllers based on colored Petri net models and the standard IEC 1131 Part II: design and implementation. *IEEE Transactions on Control Systems Technology*, vol. 7, no. 6, pp. 666-674.

[102] Huang, Y., Jeng, M., and Chung, S., 2001. Design, analysis and implementation of a real-world manufacturing cell controller based on Petri nets. *International Journal of Computer Integrated Manufacturing*, vol. 14, no. 3, pp. 304-318.

[103] Feldman, K., Colombo, A. W., Schnur, C. and Stőckel, T., 1999. Specification, design, and implementation of logic controllers based on colored Petri net models and the standard IEC 1131 Part I: specification and design. *IEEE Transactions on Control Systems Technology*, vol. 7, no. 6, pp. 657-665.

[104] Park, E., Tilbury, D. M., Khargonekar, P. P., 1999. Modular logic controllers for machining systems: formal representation and performance analysis using Petri nets. *IEEE Transactions on Robotics and Automation*, vol. 15, no. 6, pp. 1046-1061.

[105] Mahadevan, B. and Narendran, T. T., 1993. Buffer levels and choice of material handling device in Flexible Manufacturing Systems. *European Journal of Operational Research,* vol. 69, no. 2, pp. 166-176.

[106] Stecke, K. E. and Solberg, J. J., 1981. Loading and control policies for a flexible manufacturing system. *International Journal of Production Research*, vol. 19, no. 5, pp. 481-490.

[107] Fan, I. S. and Sackett, P. J., 1988. A PROLOG simulator for interactive flexible manufacturing systems control. *Simulation*, vol. 50, no. 6, pp. 239-247.

[108] Agnetis, A., Pacciarelli, D. and Rossi, F., 1996. Lot scheduling in a two-machine cell with swapping devices. *IIE Transactions*, vol. 28, no. 11, pp. 911-917.

[109] Chen, J. and Chung, C.-H., 1996. An examination of flexibility measurements and performance of flexible manufacturing systems. *International Journal of Production Research*, vol. 34, no. 2, pp. 379–394.

[110] Wu, S. D., Storer, R. H. and Chang, P.-C., 1993. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers and Operations Research*, vol. 20, no. 1, pp. 1-14.

[111] Fang, J. and Xi, Y., 1997. A rolling horizon job shop rescheduling strategy in the dynamic environment. *International Journal of Advanced Manufacturing Technology*, vol. 13, no. 3, pp. 227-232.

[112] Bierwirth, C. and Mattfeld, D., 1999. Production scheduling and rescheduling with Genetic algorithms. *Evolutionary Computation*, vol. 7, no.1, pp. 1-17.

[113] Wong, T. N., Leung, C. W., Mak, K. L., and Fung, R. Y. K., 2006. Integrated process planning and scheduling/rescheduling – an agent-based approach, *International Journal of Production Research*, vol. 44, no. 18-19, pp. 3627–3655.

[114] Montgomery, D. C., 1997. Design and analysis of experiments, Fourth edition. *John Wiley & Sons*, New York.

[115] DiCesare, F., Harhalakis, G., Proth, G. M., Silva, M. and Vernadat, F. B., 1993. Practice of Petri nets in manufacturing. *Chapman & Hall*, London.

[116] Laftit, S., Proth, J.-M., and Xie, X.-L., 1992. Optimization of invariant criteria for event graphs. *IEEE Transactions on Automatic Control*, vol. 37, no. 5, pp. 547-555.

[117] Song, J.-S. and Lee, T.-E., 1998. Petri net modeling and scheduling for cyclic job shops with blocking. *Computers and Industrial Engineering*, vol. 34, no. 2, pp. 281-295.

[118] Ahuja, R. K., Magnanti, T. L. and Orlin, J. B., 1993. Network flows: Theory, algorithms and applications. *Prentice Hall*, New Jersey.

[119] Campos, J., Chiola, G., Colom, J. M. and Silva, M., 1992. Properties and performance bounds of timed marked graphs. *IEEE Transactions on Circuits and Systems*, vol. 39, no.5, pp. 386-401.

[120] Murata, T., 1989. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580.

[121] Tuncel, G. and Bayhan, G. M., 2007. Applications of Petri nets in production scheduling: a review. *International Journal of Advanced Manufacturing Technology*, vol. 34, no. (7-8), pp. 762-773.

[122] Hillion, H. P. and Proth, J.-M., 1989. Performance evaluation of job-shop systems using timed event-graphs. *IEEE Transactions on Automatic Control*, vol. 34, no. 1, pp. 3-9.

Appendix A:

Introduction to Petri Nets

A PN is a general purpose graphical and mathematical tool, especially suited to model discrete event dynamic systems (DEDSs). Its graphical representation is extremely easy to understand. Hence, it provides a very powerful medium of communication between theoreticians and practitioners (or modelers and users). As a mathematical tool, a PN model can be described by a set of linear algebraic equations or other mathematical models that reflect the behavior of the system. This enables the formal analysis of a given model. Compared to other DEDS modeling methodologies, like Markov Chains and Queuing Networks, PNs provide a convenient framework for describing and analyzing DEDSs, are relatively easy to learn and utilize, can be automatically analyzed by the use of software, can efficiently model many of DEDS features, and can serve as a ready simulation tool for the system.

Basically, PNs are directed graphs described by three types of objects; places, transitions and directed arcs connecting places to transitions and transitions to places [120]. Places represent conditions and are depicted by circles. Transitions represent events and are depicted by bars or boxes. Input and output places to transitions represent the pre-conditions and post-conditions of events, respectively. Tokens appear in places as small solid dots to indicate whether a condition associated with a place is true or false.

The number of tokens in a place describes the local state (or *marking*) of the place [115] and the marking of the whole net describes the state of the modeled system.

A weight of an arc in the PN represents the capacity of that arc, or the maximum number of tokens that can simultaneously flow through this arc. The present work will only consider PNs that have unity-weighted arcs, which are called *ordinary* PNs. In order to simulate or model the dynamic behavior of a system, the PN has to be executed (or change its state) by *firing* its transitions. The execution of the PN is controlled by the number and distribution of tokens in the places as follows [120]:

i. A transition is *enabled* if each input place to this transition is marked by at least one token (ordinary PN).
ii. An enabled transition may or may not fire, depending on the occurrence of some *external event* (if any) that may be associated with that transition.
iii. The firing of an enabled transition removes one token from each input place and adds a token to each output place of this transition.

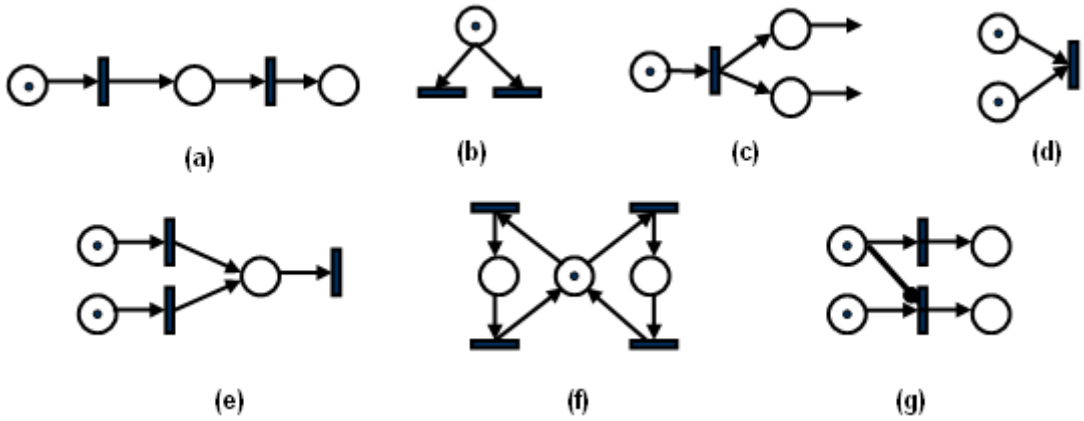
Algebraically, the structure and the behavior of a PN can be defined by means of the *incidence matrix* (IM) and the *state equation* of the net, respectively. The incidence matrix of a net is a matrix of integers indexed by the number of transitions and places of the net. The entry of the matrix is given as the difference between, the weight of the output arc from a transition t to a place p, and the weight of the input arc from p to t [120]. In the case of ordinary PNs, these entry values can then either be 1, -1, or 0. The state equation of a net defines a possible future marking of the net based on a given firing sequence of transitions, the current (or initial) marking of the net, and the IM. Hence, it

can be used to determine the expected behavior of the net, and the set of reachable markings from a current marking.

A.1 Modeling Power of PNs

The power of system modeling with PNs lies in their ability of representing many characteristics of DESs (Figure A.1). These include sequential execution, choice (or conflicts), concurrency, synchronization, merging, mutual exclusion, and priority. Priorities have been sometimes modeled in PNs by the use of inhibitor arcs; a marked place preventing an output transition from firing if it is connected to that transition with an inhibitor arc. Graphically, the ends of inhibitor arcs are depicted by circles instead of arrows, and firing the inhibited transition does not remove the token from the input place.

Figure A.1: Modeling power of PNs; a) Sequential execution, b) Choice, c) Concurrency, d) Synchronization, e) Merging, f) Mutual exclusion, g) Priority



Initially, the concept of time was not explicitly provided in the PN formalism. However, for performance evaluation and scheduling purposes of DESs, timing was

explicitly associated with PN models [120]. Timed PNs (TPNs) are defined by: i) Topological structure, which generally takes the form of basic PNs, ii) Labeling, which associates numerical values representing times with transitions and/or places, and iii) Firing rules that control the process of moving tokens around the net according to the specified times. When a TPN is transition-timed, a token is withdrawn from all the input places of the timed transition as soon as the transition is enabled. The firing process then takes the amount of time associated with the transition, after which tokens are placed in the output places of the transition. When a TPN is place-timed, tokens in timed places do not enable the output transitions unless the associated time with the place has passed. In general, when TPNs are transition (place)-timed, transitions (places) represent the system operations [115]. TPNs have also been classified into deterministic TPNs and stochastic TPNs. However, in the context of automated systems, focus has been on deterministic TPNs.

A.2 Behavioral Analysis using PNs

PNs support the analysis of many properties associated with DESs. In general, there are two types of properties that can be studied using PN models; structural and behavioral properties. Structural properties only depend on the structure of the net with no regard to its initial marking. Behavioral properties, on the other hand, depend on the initial marking of the net and they include reachability, boundedness, liveness, reversibility, safeness, and conservativeness [120]. In the current context, focus will be directed only towards some of the behavioral properties that are directly related to the scheduling and SC

problem, namely *reachability*, *reversibility* and *liveness*. The reachability problem is that of finding out if a specific marking M_n of a given net is reachable from its initial marking M_o. M_n is reachable from M_o if there exists a firing sequence of transitions that derives the net from M_o to M_n. The reachable set of a net $R(M_o)$ is the set of all possible markings reachable from M_o. In previous control literature, reachability analysis has been utilized mainly in identifying and avoiding reachable deadlock states [28]. This type of analysis has also been utilized in the production scheduling literature to identify the final state of the system that optimizes some performance criterion [121]. However, reachability analysis suffers from the state explosion problem [120]. Consequently, it has been relatively associated to systems that feature a limited size (and hence state space).

Reversibility is the ability of the system to return to its initial state. This notion is very important in the supervisory control (SC) problem and in cyclic scheduling policies that are based on periodically repeating the obtained production schedule during a given time period. A PN is said to be reversible if the initial marking (state) M_o is reachable from each marking in $R(M_o)$. This condition can sometimes be relaxed by defining some home states M', which can be reached from each marking in $R(M_o)$ [120].

Liveness is the PN property that investigates the complete absence or presence of deadlocks in the modeled DES. That is why, this PN property has been the most studied in deadlock-free scheduling and SC literature. A PN is live if from any marking in $R(M_o)$, it is possible to ultimately fire any transition in the net by progressing through some firing sequence [120]. Hence, liveness guarantees the deadlock-free operation of the modeled system, no matter what firing sequence is chosen. Because liveness can be

viewed as a very strong property in some applications, different levels of liveness have been defined for PNs. A transition in a PN is said to be:

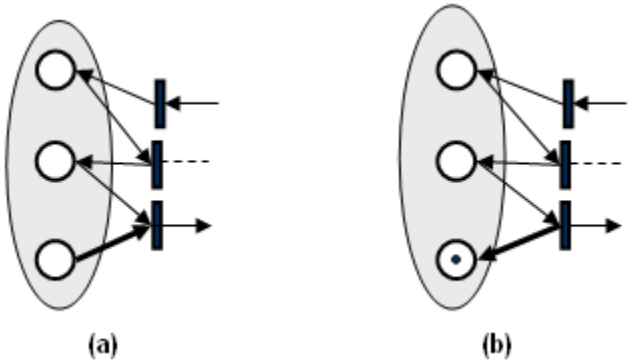
1. L_0-*live* (or dead) if it can never be fired in any firing sequence.
2. L_1-*live* (potentially firable) if it can be fired at least once in some firing sequence of the PN.
3. L_2-*live* if it can be fired at least for a given number of times in some firing sequence of the PN.
4. L_3-*live* if it can be fired infinitely often in some firing sequence of the PN.
5. L_4-*live* (live) if it is L_1-*live* for every marking in $R(M_o)$.

A PN is L_k-*live* if every transition in the net is L_k-*live*. Hence, a PN is live if every transition in the net can be fired at least once in some firing sequence from every marking in $R(M_o)$. It should be noted, however, that in the PN context, deadlock-freeness is not equivalent to, and is a weaker condition than liveness. A PN is said to be deadlock-free if at least one transition is enabled at every reachable marking in $R(M_o)$ [35]. This implicitly means that, even if some transitions in the net are dead, a PN can still be defined as a deadlock-free one. Since, in a manufacturing environment this might indicate that some operations can never be started, in the SC and scheduling of manufacturing systems literature, the PN liveness property has been the one adopted to characterize the deadlock-freeness of the modeled systems.

In a PN, a siphon is a subset of places, where any input transition to this subset is also an output transition (Figure A.2). A trap is a subset of places, where any output transition of this subset is also an input transition [120]. A *basic* siphon (trap) cannot be represented

as a union of other siphons (traps). A *minimal* siphon (trap) does not contain any other siphons (traps). In previous PN literature, siphons and traps have been directly related to the liveness of PNs. Because its input transitions are a subset of its output transitions, if a siphon becomes unmarked (free of tokens), it will remain unmarked for any net evolution. Accordingly, all of its input and output transitions will be dead (L_0-*live*), and hence the PN will not be live. On the other hand, if a trap becomes marked, it will remain marked for any net evolution [115].

Figure A.2: Special net subsets; a) A siphon, b) A trap

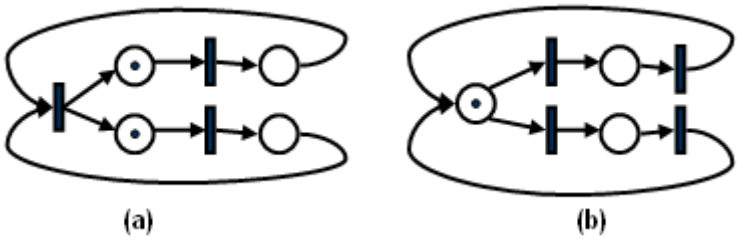


In previous literature, siphons have been considered as potential deadlocks, and the absence of empty siphons in some PN structures implied the liveness of the net [35]. Furthermore, a siphon that contains an initially marked trap will never be empty, and hence will not cause a deadlock. A considerable portion of the PN literature related to deadlock-free scheduling and control has been associated with analysis of different types of siphons.

A.3 Common PN Structures

Throughout the literature, different subclasses of PNs have been defined based on the underlying structure of the net. The most common of these subclasses are *marked graphs* (MGs) and *state machines* (SMs). A MG is an ordinary PN in which each place has exactly one input transition and one output transition (Figure A.3(a)). A SM is also an ordinary PN in which each transition has exactly one input place and one output place [120]. Aside from siphon analysis, the characterization of liveness in SMs and MGs in their basic forms can be associated to the structure of the net. Because in a SM, firing a transition moves only one token from one place to another place, a SM is live if it is *strongly connected* and M_o marks at least one place in the net (Figure A.3(b)). A PN is said to be strongly connected if there exists a directed path from each node (place or transition) in the net to every other node in the net.

Figure A.3: PN subclasses; a) Marked graphs, b) State machines



On the other hand, a MG is live if M_o marks at least one place in each *circuit* of the net. A circuit is a directed path that goes through the PN from one node back to this node such that no other nodes are repeated [122]. Accordingly, a MG is live if the net structure obtained by deleting all the places marked by M_o contains no circuits. Furthermore, a *live*

MG is also reversible [119]. Because of their potential modeling power and the ease of their analysis, SMs and MGs have often been extended and analyzed [34, 35] to model different kinds of manufacturing systems and characterize the associated deadlock situations.

Appendix B:

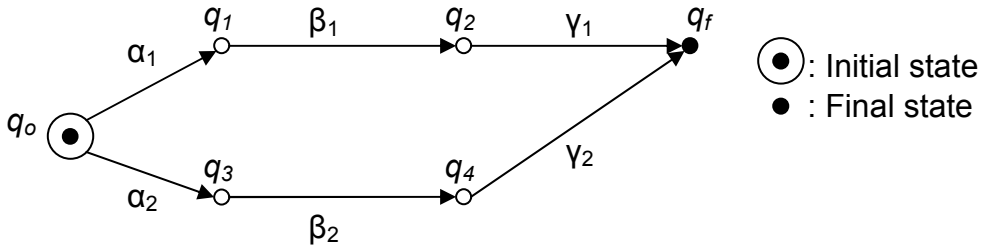
The Supervisory Control Theory (SCT)

The SCT theory was proposed by Ramadge & Wonham (RW) in 1987 [8]. The theory considered logical DES models in which times of occurrence of events are ignored and only the order in which they occur is considered. Logical models have often been employed to describe the qualitative properties of DESs, which was the main concern of the theory. In the SCT, the behavior of a plant is described by the set of sequences (or strings) of possible events, which forms a language constructed with the alphabet of the events. It further assumes that the plant generates controllable and uncontrollable events spontaneously. The behavior of this plant can be controlled by a supervisor which can disable some controllable events based on the string of events generated by the plant. The objective of the SCT is to eventually design a supervisor (or supervisors) that makes the plant behave according to the given specifications in a deadlock-free manner.

The first step in applying the SCT is to model an automaton: $G = (Q, \Sigma, \delta, Q_m, q_o)$, which is called a generator of the DES, where Q is the set of states q, Σ is the set of event labels called the alphabet, δ is the transition function that defines the resulting state upon the occurrence of an event, Q_m is the set of final (or marked) states and q_o is the initial state. The set of events Σ is partitioned into two disjoint sets; Σ_c, the set of controllable events, and Σ_{uc}, the set of uncontrollable events. Controllable events are events that can be prevented from occurrence, while uncontrollable events are those which are

permanently enabled. Along with Σ, a set Σ^* is used to denote the set of finite strings of elements in Σ. When Q is finite, G can be described as a finite-state automaton and can be represented by a directed graph. In this graph, the nodes are the states in Q, the arcs are the transitions defined by the function δ, and the set of labels for the arcs are the events in Σ (Figure 2.1). The set of all physically possible paths that initiate from q_o is denoted by L, the language (behavior) over the alphabet Σ. Another language L_m is also defined to represent the sequences that can lead to completed tasks that correspond to Q_m. To model interactions between resources in the system, a synchronous (shuffle) product of the generators of the interacting resources is obtained, namely: $G = G_1 \parallel G_2$. The behavior of G (the plant) is then determined by the concurrent behavior of G_1 and G_2.

Figure B.1: A directed graph for a generator



The final step is to define a supervisor f of G, to enable or disable controllable events by means of control inputs γ, based on the language generated by G. The supervisor acts as a map of inputs over the language that specifies for each possible string of generated events the control input needed to be applied. When a DES is supervised by a supervisor f, it obeys the additional constraints forced by f, and the behavior of the system becomes the supervised language L_f. Because of the presence of uncontrollable events, often the language L_f (or its equivalent language K) is not fully controllable. In which case, the

problem becomes one of finding the largest controllable subset of language K (supremal sub-language). This sub-language $K^{\uparrow}$ requires the least (or optimal) amount of control action to preserve the restrictions of language K.

A supervisor has also been defined, alternatively, as an automaton $T = (\Sigma, X, \xi, x_o)$ and a feedback function φ, where φ is a map of the set of control inputs over the set of supervisor states X. G then plays the role of the plant, T functions as an observer, and φ as the feedback. The outputs of G drive the state transitions of T, which determines the next control input γ on G through feedback φ [8, 9, 10].

Appendix C:

Operational Data for Illustrative Examples

Table C.1: Processing routes and times for the '6J x 3M' numerical example

Jobs	Operations Machine (Processing Time)		
	1	**2**	**3**
1	M2(25)	M3(37)	M1(38)
2	M1(39)	M2(25)	M3(25)
3	M3(33)	M2(22)	M1(21)
4	M3(24)	M1(24)	M2(37)
5	M3(29)	M1(24)	M2(40)
6	M1(27)	M2(27)	M3(28)

Table C.2: Processing routes and times for the '5J x 5M' comparison problem

Jobs	Operations Machine (Processing Time)				
	1	**2**	**3**	**4**	**5**
1	M1(8)	M2(8)	M5(9)	M4(5)	M3(2)
2	M1(9)	M2(4)	M3(9)	M4(6)	M5(3)
3	M1(8)	M2(8)	M5(9)	M4(5)	M3(2)
4	M1(9)	M2(4)	M3(9)	M4(6)	M5(3)
5	M1(8)	M2(8)	M5(9)	M4(5)	M3(2)

Table C.3: Processing routes and times for problem ‘4J x 3M’

Jobs	Operations Machine (Processing Time)		
	1	**2**	**3**
1	M1(40)	M2(100)	M3(36)
2	M2(45)	M1(65)	M3(98)
3	M1(212)	M2(73)	M3(32)
4	M3(55)	M2(65)	M1(35)

Table C.4: Processing routes and times for problem ‘ft06’

Jobs	Operations Machine (Processing Time)					
	1	**2**	**3**	**4**	**5**	**6**
1	M3(1)	M1(3)	M2(6)	M4(7)	M6(3)	M5(6)
2	M2(8)	M3(5)	M5(10)	M6(10)	M1(10)	M4(4)
3	M3(5)	M4(4)	M6(8)	M1(9)	M2(1)	M5(7)
4	M2(5)	M1(5)	M3(5)	M4(3)	M5(8)	M6(9)
5	M3(9)	M2(3)	M5(5)	M6(4)	M1(3)	M4(1)
6	M2(3)	M4(3)	M6(9)	M1(10)	M5(4)	M3(1)

Appendix D:

Structures of Robot Arm Control Programs

- Structure of *main program* (program 0):

10 If INP1= 1 &INP2 = 0 &INP3= 0 &INP4 = 0 &INP5= 0 &INP6 = 0, Call Sub-program 1.

20 If INP1= 0 &INP2 = 1 &INP3= 0 &INP4 = 0 &INP5= 0 &INP6 = 0, Call Sub-program 2.

:

:

440 If INP1= 1 &INP2 = 0 &INP3= 0 &INP4 = 0 &INP5= 0 &INP6 = 0, Call Sub-program 44.

- Structure of *Sub-program 1:*

10 Set OUT1 = 0 (Set the value of output line 1 to 0 to indicate that the robotic arm has been acquired).

20 Go to the position of $I_{A.}$

30 Pick-up Job A.

40 Go to the position of M_1.

50 Drop-off Job A.

60 Return to home position.

70 Reset OUT1 = 1 (robot arm at home position).

80 RETURN to Main Program.