A Cooperative Dispatching Approach for Scheduling in Flexible Manufacturing Cells

by

Ahmed W. El-Bouri

A thesis
presented to the Faculty of Graduate Studies
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in
Mechanical and Industrial Engineering

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

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

A Cooperative Dispatching Approach for Scheduling in Flexible Manufacturing Cells

BY

Ahmed W. El-Bouri

A Thesis/Practicum 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

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

This thesis is dedicated to my parents, Wahbi El-Bouri and Obaida Kanaan, and to my lovely wife Hana, and our two children May and Wahbi.

Abstract

A cooperative dispatching approach is proposed for scheduling a flexible manufacturing cell (FMC) that is modeled as a m-machine flowshop. Many of the current flowshop scheduling heuristics and algorithms are either inflexible or based on assumptions that are overly restrictive for the highly automated FMCs. Priority dispatching rules, on the other hand, are more flexible but their reliance on local data can frequently result in mediocre schedules, particularly in the case of flowshops. Cooperative dispatching combines heuristic qualities and the flexibility of dispatching rules in a distributed scheduling procedure that employs more global and real-time data to support dispatching decisions at the machines. A dispatching selection at any machine is reached collectively after consultation, through agents operating over a local area network, with the other machines in the cell. The consultation is initiated every time a machine needs to make a loading decision, and it takes the form of a poll that seeks a consensus regarding which of the candidate jobs should be selected, taking into consideration the performance criterion. Neural networks are available to assist the machines in formulating their replies when polled. The cooperative dispatching approach was tested in computer simulations and compared to traditional dispatching rules, for cases of both static and dynamic job arrivals. It performed consistently better than leading dispatching rules for three different criteria and in three routing configurations. Cooperative dispatching was also observed to be less sensitive than other dispatching rules to the amount of part overtaking permitted in the intermediate buffers, an issue that is relevant to FMCs which may have particular in-process buffer selection constraints stemming from automation hardware restrictions.

Acknowledgements

I am very grateful to my supervisor, Prof. Subramaniam Balakrishnan, without whose advice, guidance and support this work would not have been possible. The time and energy devoted by Prof. Balakrishnan to this project, and his practical insights, are profoundly appreciated. It was also my fortune to have the benefit of the valuable advice of Prof. Neil Popplewell. Prof. Popplewell's knowledge and experience, as well as his highly motivating encouragement, were instrumental to this thesis. In addition, I would like to thank Mr. Ken Tarte for his indispensable assistance in setting up and running the equipment that was used in the experimental part of this work. Finally, I wish to acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

Contents

List of Figures

List of Tables

Nomenclature

n	number of jobs to be scheduled
m	total number of machines in the cell
k	machine number
$p_{i.k}$	processing time for job i on machine k
h_i	holding cost/unit for job i
t_i	tardiness cost/unit time for job i
d_i	due date for job i
Δ_{ik}	operation due date for job i on machine k
C_i	completion time for job i
S	a schedule of n jobs
S^*	an optimal schedule ($S^* \in S$)
s	number of machine where job dispatching is considered.
y	index number of position in the buffer queue.
Γ_k	set of jobs waiting at machine k's buffer
Γ_s	set of jobs available for dispatching
n_{Γ_s}	number of jobs waiting in the buffer at machine s
Γ_k^y	index number of y^{th} job in sequence at machine k's buffer
x	index number of job considered for dispatching ($x = \Gamma_s^y$)
β_x	set of machines remaining on job x's route
β_x^i	the i^{th} machine in job x's route starting from machine s
n_{β_x}	number of machines that job x still has to visit.
μ_k	the position of machine k in the route for job x (β_x)
Ω_k	set of jobs that have yet to visit machine k
Φ_s	set of machines visited by the jobs in Ω_k
R_k	ready time for machine k

Ψ_k	set of jobs waiting in the buffer at machine k.
$\lambda_{x,k}$	earliest finishing time for job x on machine k
U_k	busy/idle status of machine k
a_k	number of the job currently processing on machine k
u_k	processing time remaining for current job on machine k.
η	most recently dispatched job in the cell
ξ	list of scheduled events in the cell
λ_q	expected number of new job arrivals per unit time
t_r	time between arrivals of two consecutive jobs
SC_{ky}	sequence cost when job Γ_k^y is the initial job on machine k
G	set of candidate jobs for cooperative dispatching
ρ_x	cost of selecting job x for dispatch
L_k	processing time for current job on machine k
W_k	weight factor for machine k
v_k	the minimum sequence cost for machine k
D_k	machine k's dispatching candidates
y^*	position in queue occupied by the winning candidate
m_a	value of performance measure for method under evaluation
m_a	value of performance measure for reference method
PR	performance ratio
q	input pattern to neural sequencer
G_q	target output for pattern q
O_q	actual output for pattern q
e_q	positioning error for pattern q
TD	displacement error of jobs in a sequence.

Chapter 1

Introduction

1.1 Background

Modern manufacturing operates in highly competitive environments that demand reduced costs and low lead times. Approximately 60 to 80% of the manufacturing of discrete parts involves mid-variety, mid-quantity products. The emergence of group technology (GT), coupled with advances in reducing set-up times, has allowed lower lead time and reduced levels of in-process inventory for this production category. GT emphasizes the production of like products in dedicated manufacturing cells. Flexible manufacturing cells (FMCs) implement this GT concept in an environment that is characterized by high automation. A FMC is a collection of machines that are capable of producing families of parts bearing similar production characteristics. Very short set-up times are made possible for the products by the high level of automation and tool-change capability. Part movements in the cell are performed by using automated material handling, such as an Automated Guided Vehicle (AGV) or robots. A Flexible Manufacturing System (FMS) is a collection of FMCs supported by an inter-cellular handling system.

Due to its fast product changeover, a FMC allows the simultaneous processing of different parts in small lot sizes. This simultaneous processing has the advantage of decreasing the work-in-process (WIP) compared to when the parts are processed

in larger lots. In addition to the reduced WIP, simultaneous processing accounts for increased machine utilization, a lower flowtime, and reduced storage capacity requirements (Duffie and Piper [1]). The disadvantage of simultaneous processing is the greater complexity of scheduling.

A FMC is controlled by one or more computers that normally operate under hierarchical control. In fully centralized control, a computer acts as the cell's supervisor to communicate directly with each of the cell's components, as illustrated in Figure 1.1(a). These components are the machines, the material handling system, sensors and other control devices. The supervisor obtains information from the units, and sends appropriate commands to the individual devices in order to control the activities in the cell. The supervisor is also responsible for tracking and controlling all the part movements in the cell, dispatching part programs to the machines, and responding to faults that may occur in the cell's operations. The tasks and responsibilities for the supervisor in a centralized control system increase in complexity as the size of the cell and the number of parts it processes grows.

Centralized control systems favor fixed schedules that are stored in the system and implemented directly by the hardware. When a schedule needs to be modified or updated, the supervisor must collect all the pertinent information from the cell and generate a new and efficient schedule in a very short period (usually a matter of seconds). In such situations, hierarchical systems are at a disadvantage because of the great amount of information that needs to be collected, analyzed and processed quickly. De-centralized control enables faster rescheduling and better flexibility to accommodate variable scheduling demands.

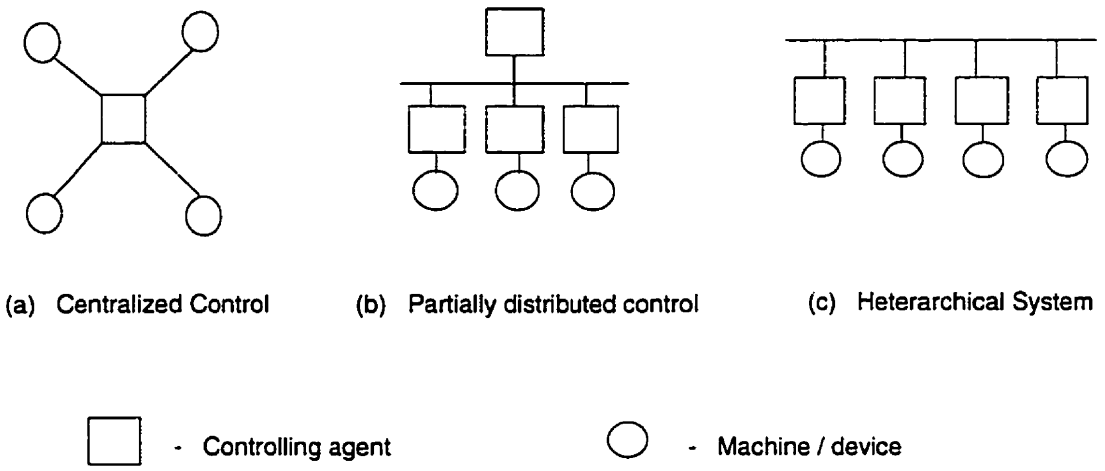


Figure 1.1: Control architectures in manufacturing [1].

At the opposite end of the spectrum to hierarchical control is the non-hierarchical system. Duffie and Piper [1] refer to the latter as a heterarchical system. In the heterarchical system, control is distributed completely and there are no supervisor/subordinate relationships. The control of part processing is achieved by communication and cooperation between the machines and devices in the cell without a central supervisor, as illustrated in Figure 1.1(c). Between the two extremes of fully centralized and fully distributed control lie hybrids of the two systems, an example of which is seen in Figure 1.1(b). In a hybrid system, control is distributed to the entities in varying degrees of de-centralization, as shown by the scale used in Figure 1.2. Existing manufacturing systems currently have control architectures that are closer to the centralized system [1]. The reason that this architecture is preferred is due mostly to the ease of its hardware implementation.

Hierarchical control, nevertheless, has numerous disadvantages resulting from

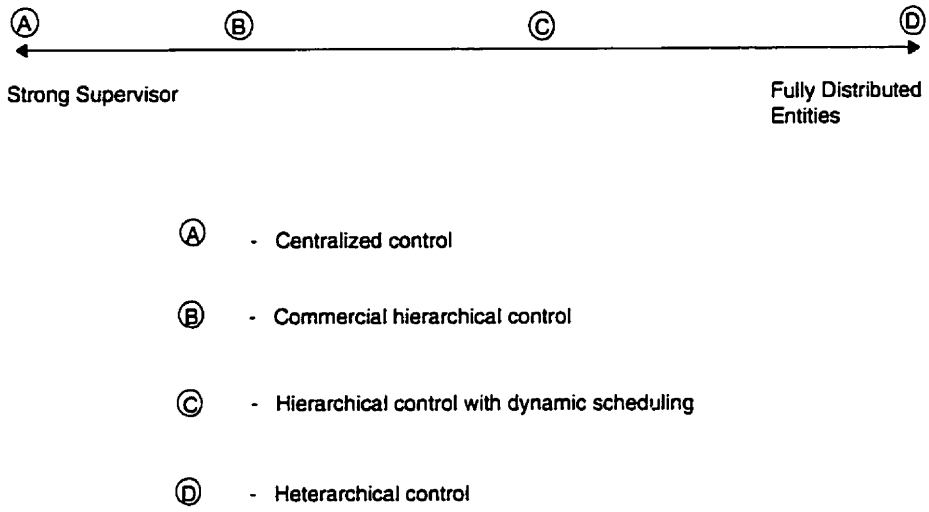


Figure 1.2: Category scale for control architectures [1].

the non-modularity of its architecture. A lack of modularity means that changes, or even minor modifications in the cell, require significant effort to modify the cell's control system. In addition, the software to run hierarchical systems is extensive, complex and costly to develop and maintain. Therefore, a manufacturing cell that is controlled hierarchically is not a very flexible one in changing environments. As an example, if the machine configuration in a cell is to be modified, then large amounts of software may have to be rewritten to incorporate the modification. On the other hand, in a non-hierarchical system, the localized nature of the control requires software that is less complex. In addition, the software is duplicated in the modules, making it more convenient to perform any updates or modifications. As far as scheduling in a FMC is concerned, the localization of information provided by modularization leads to reduced problem complexity. This, in turn, permits more efficient dynamic scheduling in the cell.

1.1.1 Scheduling Flexibility

Scheduling flexibility describes the ability to implement the best schedule possible for the current jobs and hardware setup (in the FMC) without the need for selecting, modifying or re-writing software to accommodate particular configurations and job routings. There are several issues that contribute to scheduling flexibility. Two have particular interest. The first concerns the adaptability to changing objectives and priorities. Scheduling and job prioritization are controlled by software in a FMC. If the scheduling's objectives are changed, then the software should be capable of meeting the new objectives without having to be modified. Also, disruption in cell operations, such as rush orders, reworks, machine failure etc., mean that, in reality, the cell operations are dynamic in nature. Schedules have to be updated frequently, or reworked completely to remain valid under dynamic conditions.

The second matter of interest is that raised by the high levels of automation found in FMCs. This automation may impose constraints on a cell's activities that would not be found in a non-automated system, or in one with low automation. The main constraint of interest here is the type of buffers that hold the WIP for each of the machines in the cell. Specifically, the use of robotic handling requires that parts be located at fixed locations and orientations. This means that the buffer must not only hold the parts but it has the task of delivering them to the robot's gripper at the desired pick-up point and in the correct orientation. The least costly and simplest buffers employ gravity-feed. Such buffers, however, normally restrict the order in which the parts can be processed on the machine to a first-in, first-out (FIFO) order. On the other hand, if an arbitrary selection from the parts waiting in the buffer is to be allowed, then a more complex buffer, such as a carousel, needs

to be used. In addition to being costlier, these latter buffers also require more space and software. Nevertheless, buffers that allow part selection are more desirable from the viewpoint of scheduling because they improve machine utilization and lower the WIP. It is conceivable that economic and technical constraints may result in a FMC having some buffers constrained to FIFO queues, along with others that permit a selection from the queue. Therefore, a flexible scheduling system should be effective for FMCs ranging from those that have strictly FIFO buffers to those that have all their buffers permitting a selection. To use terminology from scheduling theory, the sensitivity of the scheduling system to the amount of permissible 'part overtaking' should be minimal.

1.1.2 Jobshops and Flowshops

A FMC is actually a highly automated jobshop. In a jobshop, each part visits the machines according to a job 'route' that defines the sequence of operations necessary to complete the part. Workpieces may start and end their routes at any one of the machines. There are no restrictions on which machine a job can visit next after completing an operation on one of the other machines in the shop. On the other hand, a flowshop is a jobshop having a uni-directional flow restriction. A job may enter the flowshop at any machine, but the machines it can visit next are limited to only those downstream of the direction of the part flow. Specifically, if there are m machines that are numbered from 1 to m, then a job cannot move from one machine to another machine which has a lower number. Figure 1.3 depicts the general flowshop. When it is required that each part visits every one of the m machines in a (m-machine) flowshop, then that flowshop is called a pure flowshop.

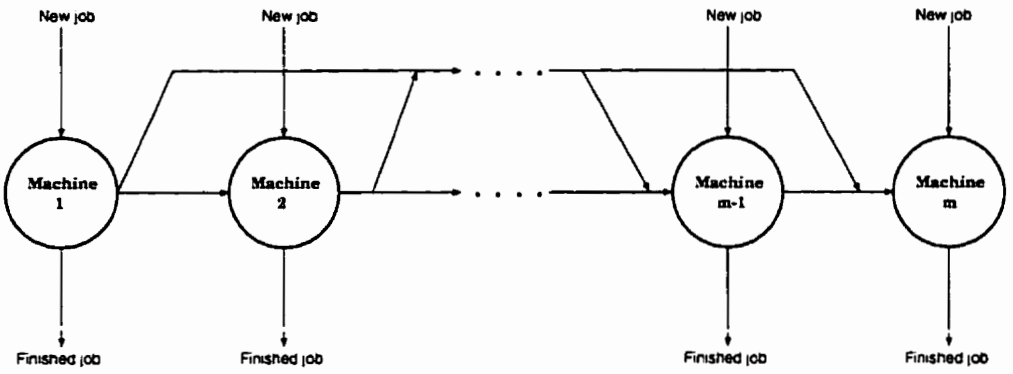


Figure 1.3: A general, m-machine flowshop.

The flow of parts in jobshops and flowshops is based on the job routings. A job enters the shop and waits in a queue at the buffer for the machine needed to process its first operation. When that operation is completed, the workpiece is transferred to wait in the queue at the buffer for the machine needed for its second operation, and so on. At any point in time, a jobshop or flowshop is likely to have jobs waiting in queues at the buffers for the machines. Obviously, instances will occur where a machine is idle because there are no waiting jobs, or because the part just completed cannot be moved to the next buffer which is filled to capacity. The former case is called 'machine starvation', while the latter is termed 'machine blockage'.

1.1.3 Dispatching Rules

Scheduling for jobshops and flowshops is a complex activity in view of the very large number of possible schedules. Finding an optimal schedule may involve the evaluation (whether directly or indirectly) of all the possible schedules in order to find the most efficient one. For a jobshop which processes n jobs on m machines,

the number of schedules that can be constructed is theoretically as high as $(n!)^m$. In practice, the number of feasible schedules is lower but it is still a significant proportion of $(n!)^m$. Even in the most restrictive case, namely a pure flowshop with no part overtaking, a total of $n!$ different schedules is possible. Mathematical techniques are available that implicitly enumerate all the possible schedules to find the optimal one. However, they are computationally expensive when n is more than 15 to 20 jobs (given current computer technology).

The combinatorial nature of the scheduling problem makes dispatching rules a favored approach for a jobshop and many types of flowshop. A dispatching rule specifies which job, from those available in a queue, has the highest priority to be selected as the next one on a machine that has just become available. The job having the highest priority is the one that is dispatched for processing on the machine. Thus, a schedule is constructed on an 'as-needed' basis, and not a priori. Consequently, dispatching rules are well-suited for dynamic scheduling.

1.1.4 Scheduling Criteria

The scheduling problem is that of arranging the sequences for the processing of jobs in the shop in a manner that allows desired objectives to be met, as much as possible. Each job passing through the shop has several operations and each operation requires a certain 'processing time' on one of the machines in the shop. In addition, the job is to be completed by a predefined time, called the due date. In the event that the due date is not met, the job is said to be tardy and a penalty is incurred that is usually a function of the tardiness.

The most commonly occurring scheduling objectives attempt to optimize one or more of the following criteria.

- Minimum makespan, where it is required to complete all the available jobs in the minimum possible time span. This criterion is relevant to static shops, where the number of jobs to be scheduled is known at the start of production and no new job arrival is allowed in the meantime.

- Minimum mean tardiness, where the emphasis is to reduce the total amount of tardiness.

- Minimum mean flowtime, which seeks to minimize the average time spent by a job in the system. This criterion helps to reduce the WIP levels.

- Minimum number of tardy jobs, where the goal is a schedule that minimizes the total number of jobs that are completed beyond their due dates.

In addition to the above criteria, many less commonly occurring ones exist, and others may also be formulated that are application specific.

More often than not, scheduling criteria are conflicting ones. For example, a schedule that minimizes the mean flowtime can be poor with respect to minimizing the mean tardiness, and vice-versa. In modern manufacturing, a WIP reduction and on-time delivery are normally co-objectives. Thus, the scheduling objective is, in reality, a combination of several criteria. There are two major approaches to deal with multiple performance criteria. The first approach is to rank the criteria in terms of importance as primary, secondary, tertiary, etc. A schedule satisfying the primary

criterion is devised. Then this schedule is adjusted as much as possible to meet the secondary criterion, without diminishing the degree of satisfaction obtained for the primary objective, and so on. The second approach is a cost-based one. A unified equivalent cost is selected to quantify the performance with respect to the different criteria. Then a single cost-based criterion is developed to represent the multiple criteria.

1.2 Research Motivation

The primary motivation for this research is based on implementing an automated scheduling system for a FMC located in the Computer Integrated Manufacturing Laboratory at the Faculty of Engineering, University of Manitoba. Although this FMC serves educational purposes, it represents real-world systems in the high degree of automation it employs. Furthermore, as a collection of autonomous sub-systems, it provides the opportunity to implement a non-hierarchical control system. The type of products made in this cell belong to a family of parts which have primary, secondary and tertiary operations that are applied sequentially. The first two machines in the cell are for the primary operations, the third machine performs the secondary operations, while the tertiary operations are performed on the last machine. The sequential manufacturing stages involved with these products give this FMC the characteristics of a flowshop.

A direct implementation of theoretical models to a highly automated cell that operates in a continuously changing environment is not simple. The difficulties are

best described by Dudek, Panwalker and Smith [2] for the case of the flowshop. They contend that there are very few real-world situations that have the characteristics of the classical flowshop assumed theoretically. The main reasons cited in [2] for the lack of industrial application of previous flowshop research are 1) overly restrictive assumptions; 2) inflexibility of the algorithms; and 3) failure to focus on the fact that real flowshops are more often dynamic (rather than static) and subject to multiple performance criteria. Although these observations relate to the pure flowshop, they are also true, to a significant degree, of other types of flow and job shops.

The second source motivating this research is automation. A FMC is not as highly automated with respect to scheduling control in comparison to the hardware. When there is a deviation from the schedule, off-line human intervention is normally needed to revise or update the schedule. One source of deviation to a current schedule is a change in the scheduling objectives. The scheduling flexibility in FMCs may be enhanced, through automation, to allow the system to quickly provide good schedules for changed objectives, multiple scheduling criteria or criteria that are unique to particular situations. The 'inflexibility' of many of the theoretical scheduling algorithms poses an obstacle to achieving a high flexibility for the scheduling component in a FMC. The flexibility that is desired for the FMCs has the following characteristics:

1. It permits adaptation to hardware reconfiguration without the need for major modifications to the scheduling software.

2. On-line adaptation is allowed whenever the scheduling criteria change.

3. A consistent performance level is provided, regardless of the types and sizes of the in-process buffers, or the predominant part routings.

4. It efficiently meets the scheduling criteria regardless of the number of different part types that are produced simultaneously in the cell.

1.3 Problem Statement

The type of FMC under consideration is illustrated in Figure 1.4. The cell contains several machines, as well as a material handling system consisting of an 'ASEA' robotic arm. The robot loads and unloads the machines, and it transports the workpieces between the machines. The activities of the robotic arm and its interaction with the machines is coordinated by means of a system of sensors, programmable logic controllers (PLCs), and the robot's controller. The PLCs monitor the signals from the sensors on the machines as well as the buffers. They send appropriate outputs to the robot's controller, which then initiates the programs corresponding to the robot's desired actions. Requests for service from the robot arm are received and dispatched by the robot's controller in an order that is generally unpredictable. The controller monitors incoming requests by means of a looping program. When a request is detected and acknowledged, the monitoring program is interrupted and the routines (programs) for servicing the acknowledged request are executed by the robot. When the requests have been serviced, the monitoring program resumes from the point at which it was interrupted. New requests arriving during the interruption may be serviced ahead or after previously waiting requests, depending where the interruption occurred in the monitoring program. Scheduling for the robot's movements is performed by an outside agent (the robot's controller)

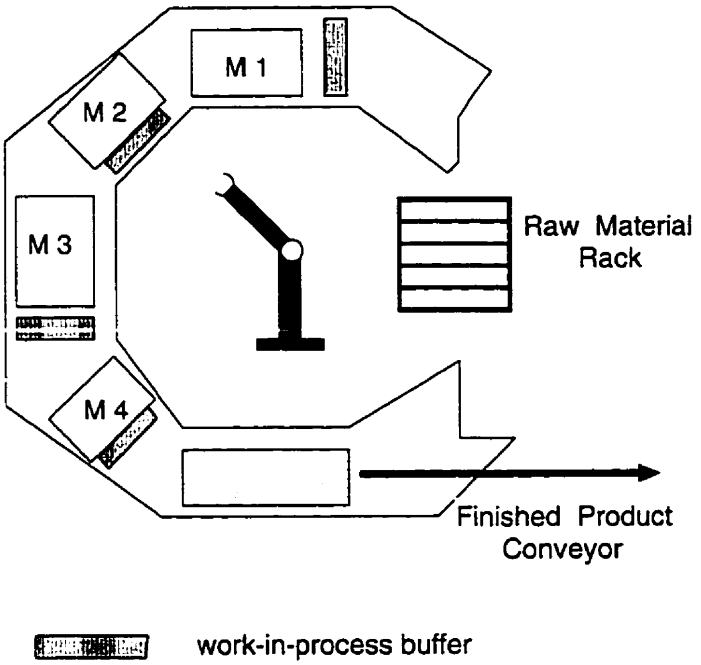


Figure 1.4: Schematic of FMC at University of Manitoba's CIM Lab.

and not by the FMC's scheduling system. The material handling system, therefore, is not controlled by a centralized supervisor (i.e. it is non-hierarchical).

The number of machines within a FMC is a function of the work envelope of the material handling device. In most FMCs that are served by a single robot, the maximum number of machines that can be accommodated realistically in the cell is between four and six. In addition, the FMC is modeled as a general flowshop. This is not overly restrictive, given that a well-designed cell that processes part families (which involve similar operation sequences) tends towards a uni-directional flow pattern. Thus, the research problem can be stated as follows.

Given a FMC similar to the one depicted in Figure 1.4 (which has the characteristics of a general flowshop), it is required to sequence the flow of jobs through the machines (or stations) in the cell such that a cost function, Z, is minimized. The cost function of interest, $Z = f(h_i, t_i)$, depends upon the holding costs (h_i) and the tardiness costs (t_i) for each one, i, of the jobs. A detailed statement of the scheduling problem and the assumptions used is given in Chapter 3.

The objective of this research is to develop a scheduling control system for this FMC that is :

1) flexible with respect to performance for different scheduling criteria;

2) consistently efficient for different routing configurations, and for different arrangements of FIFO-constrained and non-constrained intermediate buffers; and

3) implementable in an automated fashion and in a dynamic environment.

1.4 Solution Approach

The approach adopted is one that emphasizes a high degree of heterarchy in the scheduling control. This approach is facilitated by the use of a networked control system. Each station, comprising a machine and its buffer, is treated as an independent entity. The entities communicate over the network with each other and 'cooperate' in making decisions dealing with dispatching priorities for the current jobs. This anti-hierarchical approach, which promotes individuality and local deci-

sion making, makes single-machine scheduling theory an attractive tool. When all jobs have equal release times (i.e. they are all available at a given instant in time and ready to be scheduled), then single machine scheduling is basically a sequencing problem that is generally simpler to solve than multiple machine scheduling problems. The approach taken here is called 'Cooperative Dispatching'. Cooperative Dispatching uses single machine scheduling theory, with the simplifying assumption of equal release times, as the basis of the interaction between the individual entities in the cell. The results of these interactions is a series of on-line dispatching decisions that ultimately produce a final schedule. To expand the applicability to uncommon or unique scheduling criteria, a neural network is proposed for solving the single machine sequencing problems that are used in Cooperative Dispatching.

1.5 Overview

This thesis is organized as follows. Chapter 2 reviews the literature relevant to heterarchical systems and dispatching rules, with emphasis on state-dependent dispatching rules, as well as scheduling in flowshops and single machines. Cooperative Dispatching is presented next in Chapter 3, and results are given for its performance in static problems. These results are compared with those from other methods that may be used for the test problems. In Chapter 4, a novel approach is presented for sequencing jobs on a single machine by using artificial neural networks. The use of neural networks promotes flexibility by allowing performance criteria that are new or unique, and for which no algorithms are readily available. The performance of Cooperative Dispatching in dynamic flowshops is evaluated, in Chapter 5, by comparison to the more traditional dispatching rules used in similar cases. The

application of Cooperative Dispatching in the FMC at the University of Manitoba is described in Chapter 6, and results from a number of experimental trials are given. Finally, Chapter 7 provides the conclusions, and some recommendations for the direction of future research.

Chapter 2

Literature Review

2.1 Introduction

A Flexible Manufacturing System (FMS) normally has manufacturing cells linked together by material handling and information systems. Research in the area of FMS scheduling may be classified according to the level of the scheduling. This can be at the system level (the FMS as a whole), or at the cell level (individual FMCs). The scheduling problem at the system level includes tool allocation and the loading problem, which is basically the assignment of the jobs to each of the available cells (task assignment). At the cell level, the scheduling problem is confined to organizing the sequence of activities in the cell to optimally meet the desired performance criteria. In this respect, the problem at the cell level often resembles scheduling problems for jobshops and flowshops.

The focus of this survey is on the literature for scheduling at the cell level. However, under hierarchical control systems, the problem at the cell level is often influenced by decisions taken at higher levels. The literature on hierarchical scheduling in FMS is too large to be covered adequately in this survey. Only two examples are selected to illustrate approaches that attempt to address the performance of the individual cells under hierarchically controlled scheduling. The advantages of heterarchical, as opposed to hierarchical, systems is then discussed. With the modu-

larization provided by heterarchical control, the focus shifts to scheduling at the cell level. The two major approaches considered are dispatching rules and heuristics. For dispatching rules, the review concentrates on research dealing with state-dependent, dispatch rule selection. The heuristics that are discussed are those that treat the FMC as a flowshop. Finally, research on the sequencing of jobs or tasks on a single machine is reviewed briefly. This relates to the single machine sequencing optimization that is required for the algorithmic approach proposed in this thesis.

2.2 Hierarchical and Heterarchical Control

Scheduling in FMS has traditionally been part of an integrated hierarchical approach for managing a system. Problems are usually defined at an aggregated level of detail, and the information flows downward with increasingly detailed decisions taken at the lower levels. An example of such an approach is found in Stecke [3], who focuses on higher planning levels. The FMS is modeled as a closed queuing network that gives average performance levels for the aggregate input data. This information is passed to the next planning levels, where the machine groups (or pools) are identified, and jobs are assigned (loaded) for each grouping by using mixed integer programming models. The logic behind this procedure is that decisions taken at these stages enhance efficiency at the lower levels of decision, where on-line and dynamic control can be applied. The disadvantage of this approach, which is common to many hierarchically controlled systems, is that the need to finalize certain decisions before start-up limits performance under dynamic conditions.

A system for real-time operational control of a FMS under a hierarchical structure was proposed by Maimon [4]. Again, the detail of the decisions increases at each level down the hierarchy. However, there is also a feedback flow of information from the lower levels to the higher levels. This feedback enables modification of information and decision taken at the higher levels to reflect the real-time operation of the system. The highest levels generate off-line, aggregate production levels for the different part types. This is done with the aid of global databases which cover relevant information such as process plans, part routings, and machine failure rates. Scheduling control operates off-line, and it has three levels. The first level is the dynamic schedule which determines the instantaneous production rates for each part type demanded, considering the available capacities and performance criteria. The next level is the process sequencer. It is responsible for coordinating the movements in the system to enable the production rates defined in the previous level to be met. The third and last level of control is a communication level with the hardware. This level controls and monitors a machine and collects information, statistical and otherwise, for feedback to the higher levels. Results from simulations using this system show that it is capable of responding to perturbances and executing corrective decisions. However, the response is not instantaneous. The system appears suitable when many units of each part are demanded in the production mix. Its lagging response characteristic, on the other hand, makes it less effective when the mix involves a large variety and small unit demands, as is likely to be case in a FMS.

The advent of networking technology in the 1980's provided researchers with opportunities to explore non-hierarchical control in manufacturing systems. Non-hierarchical control is desirable to the degree that it makes a FMS system more dis-

tributed and dynamic, resulting in less complexity and more modularity, as well as lower development, operation and maintenance costs. Piper and Duffie [1] used the term 'heterarchical' synonymously with 'non-hierarchical' to describe a distributed control system. They identified several of the critical questions that need to be addressed in non-hierarchical systems. Furthermore, they outlined a plan of how a distributed control system would operate without centralized supervision. Piper and Duffie's concept called for each part entering the machining cell to initiate a program under a multitasked operating system. The part then communicates, through its program, with the machines it needs to visit and the material handling system in order to negotiate its way through the cell to its completion stage. The technique is highly dynamic, resulting in on-line machine assignment and self-configuration. These characteristics also allow an on-line re-configuration in the event of machine failure. A more detailed description of this cooperative scheduling approach is given in Duffie and Prabhu [5].

Shaw [6] described a method, based on the concept of cooperative problem solving, for dynamic scheduling in a non-hierarchical Computer Integrated Manufacturing (CIM) environment. The CIM had cells that cooperated by means of a network-wide, bidding scheme to schedule the jobs. The scheduling method is a two level method. In the first level, jobs are assigned to the cells. At the second level (the cell level), the jobs are scheduled within the cell. The first level scheduling is finalized through the bidding scheme and network communication. When a job has completed its current operation in a cell, that cell announces the availability of the job for its next operation. The cells in the system, including the one that makes the announcement, make bids for the job. The value of a cell's bid is the

earliest finishing time it can provide for the job for which it is bidding. In order to calculate the earliest finishing time, a cell has to reschedule its in-process jobs which include the job being bid. The cell making the most attractive bid wins the job. The scheduling of the jobs within a cell, on the other hand, is implemented by a knowledge-based planning system [7]. The performance of the bidding scheme's dynamic scheduling is compared, by using simulations, with a myopic shortest processing time (SPT) dispatching rule that is implemented through centralized control. The results reported by Shaw [6] reveal that the bidding scheme produces a significant improvement in performance. However, the effectiveness of Shaw's approach depends on the inter-cellular travel of parts because the bidding for the parts is between competing cells. In well-designed systems based on Group Technology, the inter-cellular travel is minimal. Consequently, cell level scheduling assumes greater importance when the movement of parts between cells is infrequent. That is not to say, however, that the concepts of [6] cannot be also adopted at the cell level.

2.2.1 State Dependent Dispatching Rules

A FMC has characteristics that resemble jobshops and, in many cases, general flowshops. Optimal scheduling decisions in these shops are difficult because the problem's complexity grows exponentially with the size of the problem [8]. Typically, there are $n!$ ways of sequencing n jobs waiting in queue at each machine or resource. Therefore, heuristics are often resorted to in scheduling for these shops. A popular approach employs dispatching rules. A dispatching rule uses a priority indexing scheme to determine which of the waiting jobs is processed next when a resource becomes available. Different dispatching rules use different methods for

determining the priorities. Most of the research in the performance of dispatching rules has been done for jobshops (see Conway et. al. [9]). Surveys of dispatching rules in jobshop operations may also be found in Blackstone et. al. [10] and Haupt [11]. The behavior of different dispatching rules has also been investigated in FMSs by Sabuncuoglu and Hommertzheim [12], [13], Garetti et al. [14], Ro and Kim [15] and Montazeri and Van Wassenhove [16]. The general conclusion drawn from this research is that the relative performance of different dispatching rules depends on the particularities of a system and the characteristics of the jobs, i.e. no one rule is superior for all performance criteria. An active area of research, consequently, is to determine the circumstances under which to use a given rule.

A logical approach in attempting to identify relationships between a state of the system and the effectiveness of different dispatching rules is to study how humans would make decisions under the circumstances. The behavior of humans when making scheduling decisions was compared with that of general dispatching rules by Nakamura and Salvendy [17]. Experiments were undertaken using a real-time, interactive human-FMS simulation model. Human subjects were given the task of scheduling a FMS modeled on a computer. The FMS had unlimited buffer capacities, and it represented a case of static scheduling because all the jobs were assumed ready at the start of the simulation. At each scheduling point (instants in time when a dispatching decision is required) the human scheduler was provided with pertinent information regarding the jobs available for dispatch. Experiments were done for three different scheduling criteria: minimizing the maximum tardiness, maximizing the machine utilization, and minimizing the number of look aheads. A 'look ahead' is a capability given to the human scheduler to see the consequences of decisions on

the final schedule prior to making those decisions. It aids the scheduler in dispatching. The 'look ahead' capability was included in the experiments in order to help determine its effect on performance. The results showed that, in all the test problems, the best human schedulers achieved results better than or equal to the best of eight typical dispatching rules. The effect of the 'look ahead' capability was seen to have an impact if used sparingly and only at the initial stages of the scheduling.

The results from Nakamura and Salvendy [17] underlined the ability of humans to weigh current system attributes in reaching dispatching decisions. This ability is lacking when a single dispatching rule is applied automatically at scheduling points. A number of researchers approached this issue with methods that sought to allow the selection of different dispatching rules dynamically, i.e. as situations evolved in the manufacturing system.

There are two elements involved in the dynamic selection of an appropriate dispatching rule. First, the state of the system must be represented in some fashion through identifiable attributes. Second, for any given state, knowledge must be available as to what is the most favorable dispatching rule. This problem has attracted the interest of Artificial Intelligence researchers, particularly in the areas of knowledge-based systems and neural networks.

Intelligent scheduling methods which employ knowledge-based systems generally utilize If-Then rules to reach decisions. Knowledge for the rule base is commonly gained from discrete event simulations of different states of the system. Wu and Wysk [18], Kusiak and Chen [19], and Kathawala and Allen [20], for example, con-

sidered expert systems that used rule-based inference for making scheduling decisions.

The problem of knowledge acquisition to guide the selection of a state-dependent dispatching rule is addressed by Nakasuka and Yoshida [21] with the aid of machine learning. The characteristics of an instantaneous status of a production line are captured by a set of user-defined attributes. Simulations of the production line provide examples of its instantaneous status at the points when dispatching decisions are needed. At each of these scheduling points, the current status serves as the initial condition, and the line is simulated by using one of several dispatching rules until the completion of production. The rule leading to the best final result is paired with the attributes defining the initial condition. Then this pairing is used by an inductive learning algorithm to establish a binary decision tree. The decision tree, which is basically an If-Then rule structure, can be used subsequently to establish the best dispatching rule for actual situations. The results of the method were verified by computer simulation. The amount of time needed to build the decision tree, however, was a significant disadvantage.

Shaw, Park and Raman [22] employed a similar approach to generate a decision tree. They incorporated machine learning in a rule-based environment to create a system having adaptive characteristics in the application of scheduling rules. The state of the system was described by eight attributes. Stochastic simulation was used to generate 130 training examples in order to cover a wide range for the eight attributes, together with the preferred scheduling rule for each set of attributes. Four dispatching rules, which were directed towards minimizing the mean tardiness,

were used in the learning. Experimentation showed that the scheduling directed by the decision tree performs better, as expected, than the use of a single dispatching rule.

Chiu and Yih [23] also employed induced knowledge, in this case to aid the selection of a dispatching rule at each machine every time it becomes available. They selected eight vital attributes, including the number of jobs in the system and the number of remaining operations, to describe the state of the system. A discrete event simulator was used to generate training examples in the form of pairings between different dynamic states and the corresponding preferred dispatching rule for each state. A total of four rules was considered and a multi-criterion performance measure was employed to encompass the makespan, number of tardy jobs and the maximum lateness. Chiu and Yih noted that a schedule could be described in terms of a series of dispatching decisions made at the scheduling points. A genetic algorithm was used to find good schedules from strings representing the training examples. This method reduced the time required to find solutions for the problems that provided the training examples. An incremental learning algorithm, similar to that used by Nakasuka and Yoshida [21], was then employed to extract knowledge from the solutions in the form of a binary decision tree. This tree was used to determine the dispatching rules most appropriate for the dynamic states identified during actual scheduling operations. The system also employed a performance evaluator. If the performance was deemed unsatisfactory, then the learning algorithm could modify the decision tree accordingly. Results showed that the method performed better than a static scheduling procedure based upon a single dispatching rule. The system appeared better suited to problems that have stable product mixes.

An alternative method for learning relationships between instantaneous system states and the corresponding dispatching rules makes use of neural networks. The concept is simple. Neural networks are trained to respond to an input stimulus by producing a corresponding output. When the input stimuli represent system states and the outputs correspond to dispatching rules, a trained neural network should be able to retrieve an appropriate dispatching rule when presented with an input pattern representing the system's current state.

A neural network is often used in conjunction with other techniques as part of a scheduling system. For example, Cho and Wysk [24] utilized a neural network to generate several part dispatching strategies which are subsequently evaluated in a multi-pass simulation [25]. The neural network accepted an input pattern of seven elements which defined the status of the workstation: viz. the routing complexity, performance criterion, ratio of material handling time to processing time, system congestion, machine utilization, job lateness factor and a queue status factor. The output pattern had nine elements (units), each one representing a particular dispatching strategy (or rule). The training data was accumulated from the results of computer simulations of the production system. When presented with an input pattern, the neural network responds with an output pattern that indicates the activation in each of the nine units. Each output level reflects how well the dispatching strategy is suited for the workstation status represented in the corresponding input pattern. The two most favorable dispatching strategies are selected based on the output pattern. They are then processed by a multi-pass simulator over a user-defined window of time. The strategy that better satisfies the performance criterion

is selected. Finally, the duration of the time window is important because it controls the interval between calls to the neural network. The longer is the window, the fewer are the opportunities to switch dispatching strategies. The authors found that the simulation window's ideal duration depended on the performance criterion under consideration.

A similar implementation of neural networks is described by Rabelo et al. [26], who organized the networks in a modular system. Neural networks are trained, one for each of seven performance measures, by using backpropagation to select dispatching rules for an input pattern representing the state of the system. The output of each one of these seven 'expert' networks is a ranking in order of the effectiveness of thirteen different dispatching rules. The rankings from the expert networks are directed, together with the system's state and the desired performance measure, to a gating network. The gating network releases a set of suggested dispatching rules (selected from the thirteen rules available). The gating network, which is trained by using the cascade correlation paradigm [27], gives a higher weight to the expert networks that are better able to meet the performance criteria. The authors reportedly achieved quick training and good generalization abilities in their modular neural networks system.

The methods that adopt state-dependent policies for selecting dispatching rules are subject to system 'nervousness'. Nervousness is characterized by the changing of a dispatch rule before it can have its desired effect on the system's performance. Frequent switching between different rules arises in response to vigorously changing attributes in the state of the system. Shaw et al. [22] incorporated a smooth-

ing constant which ensured adequate time for a new dispatching rule to have its desired scheduling effect. Other researchers, however, adopted more sophisticated approaches.

Ishii and Talavage [28], for example, proposed that the duration a dispatching rule is maintained should extend from the start of a transient state in the system to the beginning of the next transient state. This approach implies that a method of detecting transient states is needed. The authors proposed a function, called INDEX, for measuring a system's state at time t. The value of INDEX at time (t) is a function of the number of parts in the system, part processing times, the waiting and transportation times, and the due dates. The scheduling interval is determined by simulating the current system using FIFO over a future interval. INDEX is calculated at points within this interval. A transient state is detected by analyzing the time series data from INDEX. The scheduling interval is defined from the current instant to the instant when the value of INDEX increases (which is taken to indicate a transience). Once the scheduling interval is determined, four different dispatching rules are simulated over this interval and the rule that performs best is adopted. The scheduling algorithm was tested by using a simulation model of a FMS having four work centers, two loading and unloading stations, and three AGVs. Tests compared the performance of the transient-based scheduling interval and a multi-pass simulation method that used a constant scheduling interval against a method that employed a single dispatching rule throughout the entire manufacturing period (i.e. a single-pass algorithm). An average 5% improvement of the transient-based results were reported over the data from the single-pass algorithm which, in turn, performed better on average than the multi-pass method. Moreover, the multi-pass

method with constant scheduling intervals was less stable and it produced a more widely varying performance between problems than the transient-based scheduling interval method. It is noteworthy that, contrary to other results published in the literature for the multi-pass simulation method [25], Ishii and Talavage found this method to be somewhat inferior to algorithms based on a single dispatching rule. This difference may indicate a sensitivity of multi-pass methods to particular problem data and facility configurations.

Jeong and Kim [29] also favored simulation in a FMS as a tool for determining which dispatching rule to select and the duration it should be used. They proposed a real-time scheduling mechanism composed of three modules: a controller, a simulator, and a scheduler. The controller monitors the system's performance and updates the databases of the system's status. The controller sends a signal to the scheduler when it senses that a significant discrepancy exists between the actual and estimated performances or when a disturbance, such as a machine breakdown or a rush order, is detected. The scheduler's function is to decide which dispatching rule is to be used and when it should be used. It makes this decision after consulting with the simulator. The simulator runs each of sixteen different dispatching rules from the current time to the end of the planning horizon, and it returns the results to the scheduler. Based on the results of the simulations, the scheduler selects the best dispatching rule and relays this information to the controller for the rule's implementation. Experimentation with this scheduling system led to the conclusions, which are basically similar to those reached by earlier researchers, that a system's performance is improved by the dynamic switching of dispatching rules. Moreover, the performance is also sensitive to the method for deciding when rule switches

should be considered.

Min et al. [30] considered a competitive neural network that suggested decision rules in a multi-objective FMS. At pre-defined production intervals (for example, each day), the neural network is presented with an input vector containing the desired changes in the performance criteria for the following interval. The magnitude of the desired changes are determined by the operator in order to meet the multiple performance criteria. The input vector, therefore, contains relative data between the current values for the performance measures and the values targeted for the following production interval. The output of the neural network is a class in which the aggregate input vector is most similar to the one presented at the input. The FMS considered employs four scheduling variables. These are 1) a part's selection of a machine to move to; 2) a part's selection of a storage rack; 3) the selection of a part by a free machine; and 4) the selection of a part by the material handling system (a crane in this instance). Each decision variable uses one of between three and four different operational policies. For example, one of the policies used by the crane is to serve the closest part first, and so on. A long duration simulation is performed to generate training data for the neural network. The simulation is divided into many intervals and, in each interval, a random selection of policies for the decision variables is applied. The system's states and the performance measures are recorded for each interval. The two sets of decision rules and the differences that they produce in the performance measures between every two consecutive intervals are collected as input vectors for the neural network. The neural network is trained by the Kohonen [31] learning rule. Then, the trained network is used on-line to identify the class for the input vector representing the current states at the end of the period. A search

algorithm is used to find the closest match from the vectors in that class. Once the match is completed, the policies for the four decision variables are selected for the succeeding interval. The method was tested against a policy where the decision variables are selected randomly in each period. The results showed that the neural network is better able to respond to the operator's desired data for performance criteria related to different objectives. However, the method was not compared with static policies for the decision variables. Furthermore, it is not clear how sensitive is the approach to the stringency of the operator's demands for desired values of the performance criteria from one period to the next.

The need to control the frequency of rule switches is a negative aspect in applying state-dependent dispatching methods. A low switching frequency means less system nervousness, but it also produces a more sluggish response to a system's changes that ultimately marginalize any gains over dispatching with a single rule. An approach to deal with this problem is to use composite rules that are an amalgamation of contributions from several different dispatching rules. As a system's state changes, the relative contributions from the different rules change accordingly. Thus, the character of the rule alters gradually and not in the discrete manner that occurs in rule switching. An example of such an approach follows next.

Sim et al. [32] proposed a hybrid neural network - expert system that can be applied dynamically to make dispatching decisions. An expert system evaluates the prevailing shop conditions and determines which one of sixteen sub-networks is most appropriate for making the required dispatching decision. Each of the sub-networks is a neural network that is trained by backpropagation to make the dispatching de-

cisions under specific shop floor conditions which are defined by a job arrival rate and a scheduling criterion. The specialized neural networks are trained with data acquired from the results of simulations for ten different dispatching rules run under defined shop floor conditions. Each job that is a candidate for dispatching is represented by an input array of fourteen, 0 - 1 nodes. These nodes indicate the presence or absence of particular attributes. Furthermore, they encode the current shop conditions and scheduling criteria. The output of the neural network is a value, which lies between 0 and 1, that measures the level of priority determined by the neural network for the job represented at the input layer. After processing all jobs, the one determined by the neural network to have the highest priority is selected for dispatching. In this fashion, the dispatching rule is a composite rule defined by the relative contributions from the ten rules considered in the training stages. In comparison to techniques that switch between dispatching rules, the method of Sim et al. [32] effectively 'invents' a rule for the particular combination of job attributes and shop conditions at hand. The authors also presented results showing the superiority of the composite rule method over dispatching with any one single rule. Whether composite rules are more effective than a dynamic selection between the individual dispatching rules remains unanswered.

Although a very significant part of the research in jobshop scheduling has been devoted to dispatching rules, most real world manufacturing consists of a product mix containing a demand of individual parts in multiple numbers. Identical parts will generally possess identical attributes, resulting in a tendency for most dispatching rules to batch the production. This batching conflicts with the concepts advocated in flexible manufacturing, namely simultaneous production and a batch

size ideally equal to one unit. The need to produce in very low batch sizes assumes greater importance in flexible assembly systems, where different parts are needed simultaneously at the point of assembly. In such cases, the use of a dispatching rule like SPT gives equal priority to all the demanded quantities of a single part because all these quantities will have the same value for the selection attribute, namely the processing time. The result is that a batched production is observed in many instances when the traditional dispatching rules are employed. Analytical and heuristic methods are alternative approaches in such cases. They are usually developed for the specific type of problem at hand. Consequently, their application is usually less general than dispatching rules.

2.3 The Flowshop

The review of heuristics for scheduling in FMCs that follows covers flowshops only. This is because the FMC is modeled as a general flowshop in this thesis. Assuming that a FMC should preferably produce parts belonging to a family in accordance with Group Technology concepts, it is not unreasonable to use a flowshop model. This is because parts from a family are likely to have similar manufacturing operation sequences and, therefore, similar part routings. Thus, an FMC's layout can frequently resemble that of a general flowshop.

The most widely studied flowshop problem is a pure flowshop in which the objective is to minimize the makespan. Johnson [33] provided an algorithm that finds the optimal solution for the corresponding two-machine problem, as well as for the

three-machine problem but under specific conditions. Subsequent research has investigated methods for solving Johnson's problem when the shop has more than two machines. The number of schedules that are possible in a pure flowshop having m machines and processing n jobs is $m(n!)$. A frequently used assumption is that machines process jobs as they arrive in a first-come first-served order. This assumption reduces the scheduling problem to that of finding the best schedule from $(n!)$ possibilities. This type of problem is called a permutation schedule. In a non-permutation schedule, conversely, jobs are permitted to overtake other jobs and machines are allowed to remain idle until a specific job arrives, even though other jobs may be ready and available.

Researchers in the m-machine, n-job flowshop scheduling problem have predominantly considered the makespan criterion. Of the numerous heuristics and algorithms suggested, a simple but highly effective construction heuristic was proposed by Nawaz et al. [34]. A construction heuristic begins with a partial schedule and proceeds to expand it to a final schedule by adding jobs one at a time. Nawaz et al. first determined the sum of processing times for each job, and then listed the jobs in a non-increasing order of this value. An initial partial schedule was created next by removing the first two jobs on this list, and sequencing the two in the order that gives a minimum makespan. The next job residing at the top of the list is then removed from the list and inserted in the partial schedule. The position where it is inserted is found by considering all the possible positions it can occupy, without altering the relative positions of the jobs assigned previously to the partial schedule. The longer is the partial schedule, the greater is the number of possible insertion points that need to be examined. The job is inserted ultimately in the position

that gives the minimum makespan for the partial schedule. The jobs in the list are scheduled in a like fashion until a final schedule is obtained. A study by Park [35] concluded that the heuristic suggested by Nawaz et al. (which is commonly labeled the NEH algorithm) was the 'least biased and most effective' of sixteen heuristics tested in problems combining 15 to 30 jobs and 4 to 20 machines. The NEH algorithm's runtime is slightly longer than comparable heuristics in large (100 jobs and more) problems. However, modifications due to Taillard [36] have significantly improved the computational time.

The NEH algorithm was designed for permutation schedules. Recently, Koulamas [37] developed a construction heuristic (called HFC) that aims to minimize the makespan through repeated use of Johnson's two-machine algorithm [33] for determining a priority index for each job. Numerical experiments showed that HFC outperformed the NEH algorithm in problems where the optimal solution was a non-permutation schedule. In problems where permutation schedules are optimal, the HFC and NEH algorithms were reportedly comparable.

Initially, the main direction of flowshop research was aimed at minimizing the makespan. This was probably a consequence of the interest stirred by Johnson's work [33]. Later researchers, however, began to investigate other performance criteria in flowshops. Gupta [38], for example, described three algorithms for finding permutation schedules that minimize the mean flowtime in the n-job, m-machine flowshop. One algorithm, called MINIT, was reported to be the more effective of the three algorithms in terms of near optimality. The MINIT algorithm is based on a sequence-dominance check and an approximation based on minimizing the idle

time on individual machines.

Other work devoted to minimizing the mean flowtime with permutation schedules in flowshops include that of Miyazaki et al. [39] who utilized adjacent pairwise interchanges, as well as Rajendran and Chaudhari's three quick algorithms [40]. The latter are based on heuristic preference relations for deciding which job is appended next to a partial schedule. Computational results from Rajendran and Chaudhari [40] showed that their algorithms performed better than other methods, including the algorithms of Gupta (i.e. MINIT) [38] and Miyazaki [39]. In 1995 Ho [41] introduced a near-optimal heuristic based on sorting methods. Although this last approach was effective in finding near optimal solutions with a high level of consistency, it suffered from a significant growth in computational time as the problem size increased beyond 10 jobs. More recently, Woo and Yim [42] presented a job insertion method to minimize the mean flowtime. Their method considers all the remaining jobs when selecting one to insert in a partial schedule. Simulation experiments revealed the superiority of their algorithm over previous heuristics. However, like Ho's method [41], this superiority is achieved at the cost of CPU time which increases sharply with the total number of jobs.

The other criterion investigated for the pure flowshop is the minimization of the mean tardiness. Like the mean flowtime criterion, the mean tardiness flowshop problem is NP-hard [43]. Approaches using branch and bound techniques have been suggested by Sen et al. [44] and Kim [45] for the two-machine flowshop. The branch and bound approach is subject to a large space search and, even with the improved bounds suggested by various researchers, the method remains largely unsuitable for

problems in which the number of jobs is greater than about 15 to 20 jobs. Therefore, like other NP-hard situations, research has actively pursued heuristic solutions. Sen et al. also proposed in [44] three heuristics based on sorting techniques. Initial solutions were derived by sorting the jobs in the increasing order of their processing times (SPT), due dates (EDD) and their slack. The slack is the difference between the due date and the sum of the processing times for a job. Each of the heuristics starts with one of the three initial solutions. Then, local optimality conditions are applied to improve the initial solutions.

Kim [46] modified several heuristics developed originally for the makespan criterion in order to minimize the mean tardiness in flowshops having permutation schedules. Noting that solutions for permutation schedules may be improved easily by position exchanges, Kim considered a tabu search and a neighborhood search algorithm, as well as an adjacent pairwise interchange to improve solutions obtained by other heuristics. Kim's experiments showed that the best results came from the tabu search of Widmer et. al. [50], and an extensive neighborhood search (ENS) method. These two, however, required relatively high computational times. On the other hand, Kim's modified NEH algorithm, in which an EDD sequence is used to initially sort the jobs, gave better results than ENS in much less computational time when the resulting NEH sequence was further processed by an adjacent pairwise interchange procedure.

The minimization of a cost function that combines the mean flowtime and mean tardiness in flowshops was considered by Gelders and Sambandam [47]. They proposed four heuristics to obtain solutions to flowshop problems where the objective is

to minimize the sum of the weighted tardiness and weighted flowtime. The heuristics are construction heuristics based on a job dispatching index that calculates priorities for the jobs to be scheduled. The dispatching index identifies which of the jobs is the most expensive to hold and schedules it next in the sequence. This procedure entails obtaining the estimated unit penalty costs for each job. The authors provide a complex set of calculations for the measurement of the unit penalty cost. These calculations require the computation of lower bounds on the makespan of all the jobs in a given partial sequence, as well as the total idle time accumulated by each job on all the machines. The dispatching rule that emerges contains a factor that represents a relative measure of the lateness as well as a weighting factor for the holding costs. This dispatching rule is sensitive to these factors, and the four heuristics are different only in the method by which the dispatching index is determined. It was concluded that two of the heuristics consistently outperformed the others on the basis of the due-date's tightness and the performance criterion. One of the two heuristics appears to perform well for the mean flowtime criterion alone; the other for the mean tardiness criterion.

The permutation flowshop has also been researched by using search-based methods such as simulated annealing (Osman and Potts [48], Ogbu and Smith [49]) and tabu search (Widmer and Hertz [50]). The results are generally good but computationally expensive in comparison to the heuristics cited previously in this review.

Although the vast majority of research for flowshops is in permutation schedules, some research has been directed at non-permutation schedules. Examination of non-permutation schedules for flowshops has dwelt on dispatching rules. Kim

[46] investigated the performance of seven priority rules for dispatching, under the criterion of minimizing the mean tardiness. The rules were tested by using 1000 randomly generated problems, each having from 15 to 50 jobs, and between 5 and 25 machines in the flowshop. The due dates assigned to each job were determined by an adaptation of the method of Potts and Van Wassenhove [51] for a flowshop. The test problems covered a reasonably wide range of due date tightnesses. Kim concluded that the best performance was given by the Modified Due Date (MDD) rule, followed closely by the Apparent Tardiness Cost (ATC) and the Modified Operation Due-date (MOD) rules.

A comparative study of dispatching rules in flowshops and jobshops was undertaken by Rajendran and Holthaus [52]. The objective was to evaluate the relative performance of thirteen dispatching rules for the two shop types and dynamic job arrivals. Three of these rules were developed earlier by the same authors [53]. The study involved a number of performance criteria, including the mean flowtime, maximum flowtime, mean tardiness and the proportion of jobs that are tardy, among others. Extensive simulation experiments were conducted for a variety of machine utilization levels and due-date tightness factors. Job arrivals in the simulations were generated by using an exponential distribution for the inter-arrival times. Values of the performance measures were tabulated for each of the dispatching rules tested. The main conclusion is that the performance of dispatching rules is influenced by the job routings and shop floor configurations. Furthermore, dispatching rules that contain additional information, such as the total work content of the jobs queuing for the next operation, are more apt to simultaneously minimize multiple performance measures. The study did find, however, that the COVERT (cost over time)

dispatching rule performed better in flowshops than in jobshops. This conclusion contradicts earlier results [46] regarding the performance of COVERT in flowshops. The discrepancy may be attributed to how each of the investigators applied the COVERT rule, because the rule requires user defined parameters to estimate the lead times. Also, a noticeable omission from the dispatching rules tested by Rajendran and Holthaus is the MDD rule, a rule that has been frequently cited as being a powerful one for minimizing tardiness based criteria in flowshops.

Chan and Bedworth [54] considered the minimization of the mean flowtime in flowshops. They derived formulae for computing temporary flowtimes between pairs of jobs. Their algorithm was tested on specific configurations (where jobs can have one of several fixed routes) in what they called a modified flowshop. The tests indicated that the heuristic was preferable to using the least work remaining (LWKR) dispatching rule. In addition, the authors outlined the applicability of their heuristic to dynamic flowshops, where job arrivals at the shop are not simultaneous (i.e. unequal release times). Although the experimental results were satisfactory, they were done for static flowshops only, and included problems having only ten or fewer jobs.

2.4 Single Machine Sequencing Problems

The static scheduling for a single machine is a permutation problem. The optimal solution is one of $n!$ possible job sequences. For some performance criteria, the optimal sequence can be obtained by sorting the jobs in a particular order (for

example, SPT in the case of minimizing the mean flowtime). For other performance criteria, such as minimizing the mean tardiness, the problem is NP-hard [55]. When the scheduling criteria involve performance objectives that are NP-hard, many heuristics attempt to capitalize on special dominance properties in order to curtail the solution space. For example, Russell and Holsenback [56] used the dominance properties described by Emmons [57] in their heuristic for minimizing the mean tardiness. Other notable heuristics for mean tardiness minimization were presented by Panwalker et al. [58], Wilkerson and Irwin [59] as well as Potts and Van Wassenhove [51]. Scheduling objectives involving two (bicriterion) and multiple criteria have also been investigated. Then the main approach was to build particular heuristics for the specific combinations of the performance objectives. Examples of such instances may be found in Van Wassenhove and Gelders [60], Lin [61] and Chen and Bulfin [62].

Approaches that are more generally applicable (i.e. that are valid for any performance criteria) revolve around search-based techniques such as simulated annealing (Potts and Van Wassenhove [51]), tabu search (Armentano and Roncini [63]) and genetic algorithms (Lee and Choi [64]). Scheduling flexibility requires the ability to develop good schedules for any given performance criterion. Search based methods are flexible due to their generalized abilities, and the quality of their solutions improves with more computational time. This computational demand severely limits the effectiveness of these methods for on-line scheduling applications.

The scheduling approach proposed in this thesis is distributive in nature. It is characterized by the development of local schedules at each machine. These localized schedules, which are based on single machine sequencing, provide the framework for

the dispatching decisions that are made in order to determine which job is loaded on the next available machine. The effectiveness of the approach, therefore, rests upon the ability to find quick solutions of the single machine sequencing problem. Fast search-based methods are needed if the system is to be applicable to cases involving new, uncommon or multi-factor performance criteria. Chapter 4 in this thesis is devoted to describing a neural network that can learn and generalize relationships between jobs and the position each job should occupy in a sequence that satisfies the given performance objectives optimally. A system of such networks may then be used to quickly find near optimal sequences.

2.5 Conclusions

Two conclusions may be drawn from the preceding review of the literature on scheduling and distributed scheduling in FMSs. First, dispatching rules have a natural appeal due to their low requirements for global data, and their suitability to on-line scheduling. Secondly, a state-dependent selection of dispatching rules improves performance over the use of a single dispatching rule throughout the production period.

The concept of switching dispatching rules in response to changes in a system's attributes poses a considerable operational difficulty. That difficulty is the decision of when to switch from an existing rule, and for how long the new rule is allowed to operate before it too is reconsidered. The approach taken in this thesis employs a dispatching rule that is based on a complex cost-based evaluation of the current state

of the system as a whole. The proposed dispatching method is, effectively, the result of a cooperative decision between all the machines in a system. The rule dynamically alters priority levels for the jobs and implicitly achieves rule switching without the problem of deciding when to switch. Furthermore, no knowledge relating system states with particular dispatching rules needs to be acquired. Finally, the proposed cooperative dispatching approach avoids the 'batching' that is common to most dispatching rules when jobs having identical processing and due-date data are involved.

Chapter 3

Cooperative Dispatching

Distributed control in a FMS utilizes data that is locally available to the entities in the system. The effectiveness of the scheduling system as a whole towards meeting the performance objectives depends on the quality of the local data. The availability of a computerized network linking the individual entities enhances the local information by allowing the entities to exchange data, without the need for central supervision.

This chapter is devoted to describing and illustrating a methodology that is designed to achieve consistently good scheduling in a FMS under a distributed control environment. The methodology, called Cooperative Dispatching (CD), has been introduced cursorily in the previous two chapters. CD behaves similar to dispatching, but it focuses on boosting the quality of local data in a manner that enables overall performance objectives to be met better.

3.1 Introduction

The principle of CD and an algorithm for its implementation are presented in this chapter. A dispatching decision involves determining the priorities of the jobs waiting in queue for a machine. This prioritization is based normally on the scheduling

objective. Traditional dispatching rules, such as SPT, EDD, MDD, etc. are quick and simple methods that are commonly used to decide dispatching priorities. It has been shown [8] that, for given objectives and congestion levels, certain dispatching rules outperform others. Most research dealing with dispatching rules has been performed for jobshop environments, where the SPT rule was found empirically to be better than a host of other rules in minimizing the mean flowtime [8]. For minimizing the mean tardiness, however, a number of due-date based dispatching rules were found to be equally competent but rather sensitive to shop congestion and due-date tightness [9]. Although flowshops are specific cases of jobshops, there are indications that the conclusions with regard to dispatching rules in jobshops cannot be generalized to flowshops. For example, Kim [46] reported that the COVERT dispatching rule [8], generally acknowledged as a good one for jobshops, does not perform satisfactorily in pure flowshops. Furthermore, a rule's performance can depend on routing configurations, in addition to the congestion and due-date tightness.

Cooperative Dispatching is designed to overcome the preceding limitations by providing a new dispatching procedure that is less sensitive to the previously mentioned factors, and more flexible for different performance measures. Cooperative Dispatching is similar to traditional dispatching rules in that its purpose is to select, from a number of candidates, a job to be processed next. It does differ from traditional dispatching rules in that the dispatching decision is taken collectively, after the other machines in the cell are consulted in a unique manner. This 'consultation' involves polling the other machines for their input regarding which of the candidates available for dispatching is selected by the machine waiting to be loaded. A procedure for resolving the competing claims of the candidate jobs is employed to

find the successful candidate, which is selected and then dispatched. The processes of 'consultation' and conflict resolution which characterize this approach lead to the name 'Cooperative' Dispatching.

This chapter provides the details of Cooperative Dispatching and the algorithm proposed for its implementation. The performance of the algorithm is evaluated by using the three different routing configurations shown in Figure 3.1, with three different performance measures; viz 1) minimizing the mean flowtime, 2) minimizing the mean tardiness, and 3) minimizing the number of tardy jobs. The configurations in Figure 3.1 are selected because they represent the more interesting routing patterns. The Type I configuration represents the pure flowshop, and Type II considers the case where a machine can receive jobs from alternate machines (convergent routes). Type III is a configuration that allows more than one possible destination for jobs leaving a machine (divergent routes).

Given a flexible manufacturing system having m machines and processing n jobs, it is required to sequence the flow of jobs through the machines (or stations) in the cell such that some cost function Z is minimized.

Let

h_i = holding cost for job i per unit time.

t_i = tardiness cost for job i per unit time.

d_i = due date for job i.

n = total number of jobs available and ready for scheduling.

C_i = time of completion of all processes for job i.

B_i = max[C_i - d_i; 0].

$Z = f(h_i, t_i)$.

The function Z that is of main interest is

$$Z = \sum_{i=1}^{n} (C_i h_i + B_i t_i). \tag{3.1}$$

The following assumptions are made.

1. Jobs do not have alternate routings.

2. Set-up times are sequence independent. This means that set-up times can be included in the processing time for a job on a given machine.

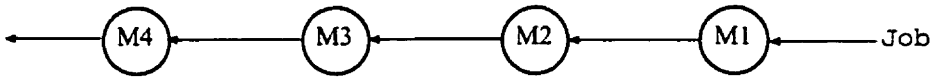
3. Pre-emption is not permitted. Once a job is started on a machine, it cannot be interrupted in favor of another job, and then resumed afterwards.

4. Inserted idle time is not allowed. A machine will not remain idle in order to wait for a particular job to arrive if there are other jobs already available and waiting to be processed.

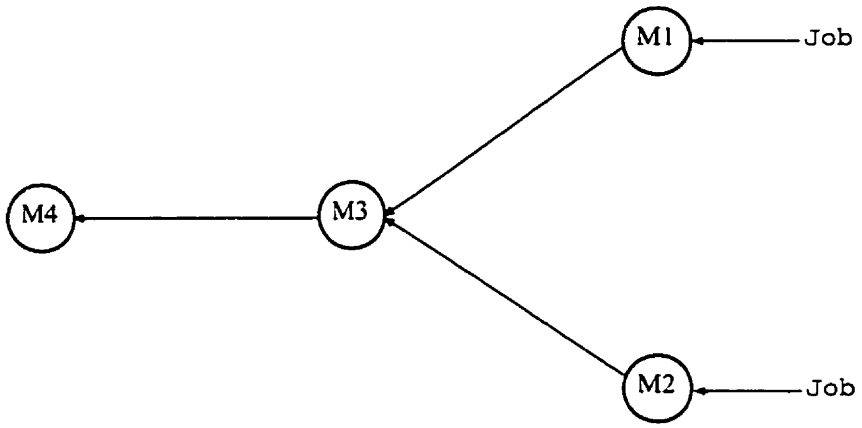
5. Buffers have unlimited capacities.

6. Some or all of the buffers may be constrained to FIFO processing of the local job queues.

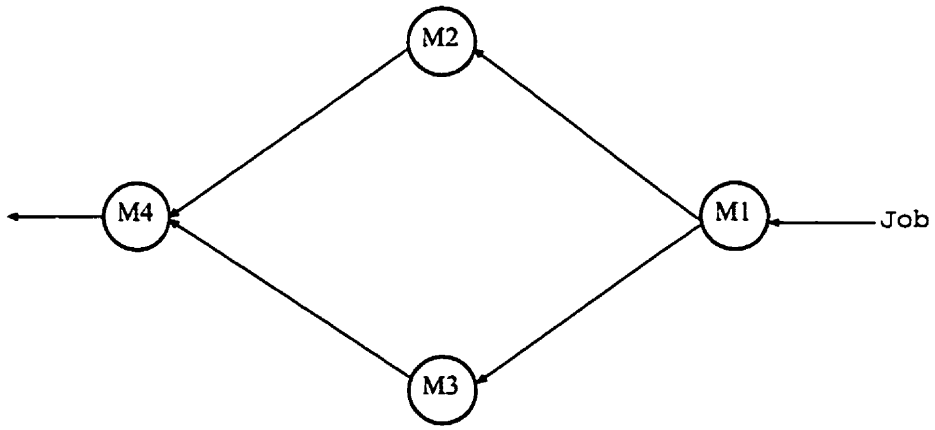
7. Parts are arriving continuously to the cell, and they immediately enter the system (i.e. the jobs have unequal release times, and the flowshop is dynamic).



Type I Configuration



Type II Configuration



Type III Configuration

Figure 3.1: Three different job routing configurations.

A static flowshop may be realized by setting the release times equal for all the jobs, thereby ensuring their simultaneous arrivals.

8. The holding cost for a job is uniform for all machines in the system.

3.2 Mathematical Model

3.2.1 Background

This section discusses the concepts behind the CD approach for sequencing the flow of jobs through a m-machine cell that may be modeled as a pure or as a modified flowshop. The approach is best described as a heterarchical [1] one in which each of the machines in the cell acts independently in attempting to optimize the performance measure. In this environment, each machine specifies the sequence that will enable it to minimize locally the performance measure under certain constraints and assumptions. The constraints relate to the earliest possible starting times (ready times) for specific jobs on the machines. The assumptions allow each individual machine to behave as though it were operating as a single machine.

The consequences of this heterarchical approach, where each machine seeks to optimize the performance measure locally and not globally, is usually a set of conflicting demands regarding the setting of the dispatching priorities. These conflicts are resolved through a cost-based methodology which aims to reconcile the local optima in a manner that will promote the 'global' solution by selecting a candidate job that is most acceptable to all the parties (machines) involved. It should be noted that not all the machines have equal footing in determining the dispatched job. In fact, each machine's 'say' in the decision is weighted by a factor that is commensu-

rate with its 'importance' at the global level. In this manner, the satisfaction of the performance criterion locally is manipulated in the direction of satisfying the performance criterion for the cell as a whole (i.e. globally). This relationship between the local and global satisfaction of the performance criteria is explored further in section 3.5.

The dispatching decision in the CD model is centered around a matrix of real numbers that is used as the basis for determining the priorities for the jobs ready for dispatching on the next available machine (machine s). This matrix, called the sequence cost (SC) matrix, has dimensions of m x n_{Γ_s}. The m is the number of machines in the cell whilst n_{Γ_s} represents the number of jobs that are immediately available for loading (i.e. the contents of the buffer at machine s). The element SC_{ky} represents what is called the sequence cost to machine k in the event that the y^{th} job in the buffer is selected for dispatch.

The **sequence cost** to a machine is defined as the value of the performance measure for a particular scheduling of the jobs on that machine. Let, for simplicity, the variable x represent the y^{th} job in the buffer at machine s (i.e. $x = \Gamma_s^y$). Then, the schedule used in determining the sequence cost for machine k is that which optimizes the performance measure on machine k, given that job x is the first in sequence. The remaining jobs that are to visit machine k are assumed to be ready and available by the time the processing of job x is completed on machine k. This assumption of equal release times is adopted solely for reducing the calculation burden. Although this simplifying assumption may seem unrealistic, it is not overly unreasonable considering that it still allows a comparison of the sequence

costs for all of the machines to be made on the basis of lower bound estimates of the performance measure. The optimal sequence of the set of jobs Ω_k on machine k, excluding job x, is determined by treating this situation as a sub-problem involving the solution of a single machine sequencing problem. Each entry in matrix SC, therefore, entails solving a separate single machine sequencing problem.

Whenever the occasion calls for a dispatching decision at machine s, CD requires that all the machines in the flowshop are polled to determine which job each machine favors. A machine is allowed to nominate two candidates, the second candidate being essentially an alternate to mitigate the negative impact in the event that its first choice is not selected. Each machine, therefore, selects its two most favored jobs, namely those two that give the least sequence costs in the SC matrix for the respective machines. Once the complete set of dispatching candidates is compiled, conflict resolution is performed with the aid of the SC matrix to determine the 'winning' job.

The main steps in the CD approach are summarized next.

1. CD dispatching is invoked whenever a machine s is available and more than one job is ready and waiting to be processed on it.
2. Sequence costs are calculated at each machine and the corresponding SC matrix is constructed.
3. By using the SC matrix, the candidate jobs for dispatching are identified. (See section 3.2.3.)
4. With the aid of the SC matrix again, one of the candidate jobs is selected for dispatching.

The above cycle constitutes what is called a 'stage' in the CD procedure. A stage is concluded after a job is selected and dispatched.

3.2.2 Constructing the SC Matrix

The cornerstone of the CD approach is the SC matrix. To explain the crucial parameters involved in constructing this matrix, the following illustrative example is introduced.

Example 3-1

An example of a pure flowshop problem with five jobs on four machines is presented in Table 3.1. All the in-process buffers for this problem are serviced according to the FIFO rule. Hence the schedule of the jobs on the machines is a permutation schedule, and dispatching occurs only at the initial machine (machine number 1).

For the purpose of explanation, it is assumed that the first three jobs (i.e. jobs 1, 2, and 3) have been dispatched already in the order of the job numbers. The resulting partial schedule is shown after the first three stages in Figure 3.2(a) in the form of a Gantt chart. A 'stage' in the construction of the schedule represents the series of calculations culminating in a dispatching decision. The partial schedule shown in Figure 3.2(a) indicates that 55 time units after the dispatch of job 1, machine 1 is available again. At this instant, the remaining possibilities are to dispatch either job 4 or job 5. The partial schedule can be updated to reflect the current status by setting the current time to zero and accordingly updating all the times. (Thus the

Table 3.1: Processing times for Example 3-1.

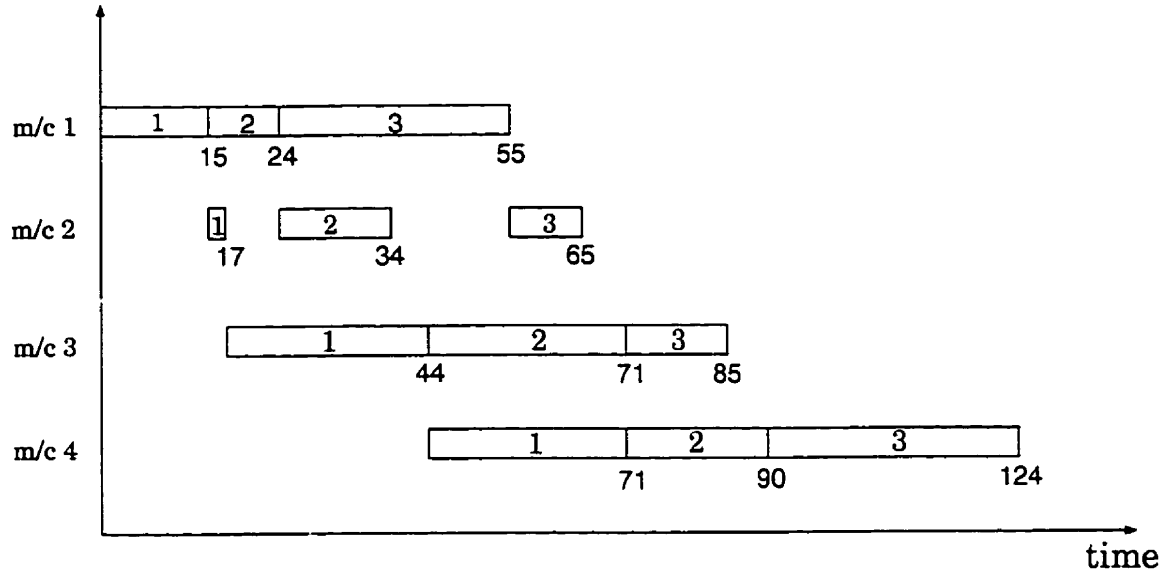
	MACHINE (m/c)						
JOB	1	2	3	4	d_i	h_i	t_i
1	15	2	27	27	192	0	1
2	9	15	27	19	159	0	1
3	31	10	14	34	124	0	1
4	33	18	21	6	95	0	1
5	15	3	3	1	73	0	1

time is advanced by the 55 units elapsed in the first three stages.) The current-time status is shown in Figure 3.2(b), where a decision is now to be made (at time zero) on machine 1. This decision involves the updated time data (i.e. the modified due dates) for the two jobs waiting to be loaded at machine 1, as shown in Table 3.2. Note that as machine 1 is the only machine in Example 3-1 where dispatching takes place, the variable s equals one.

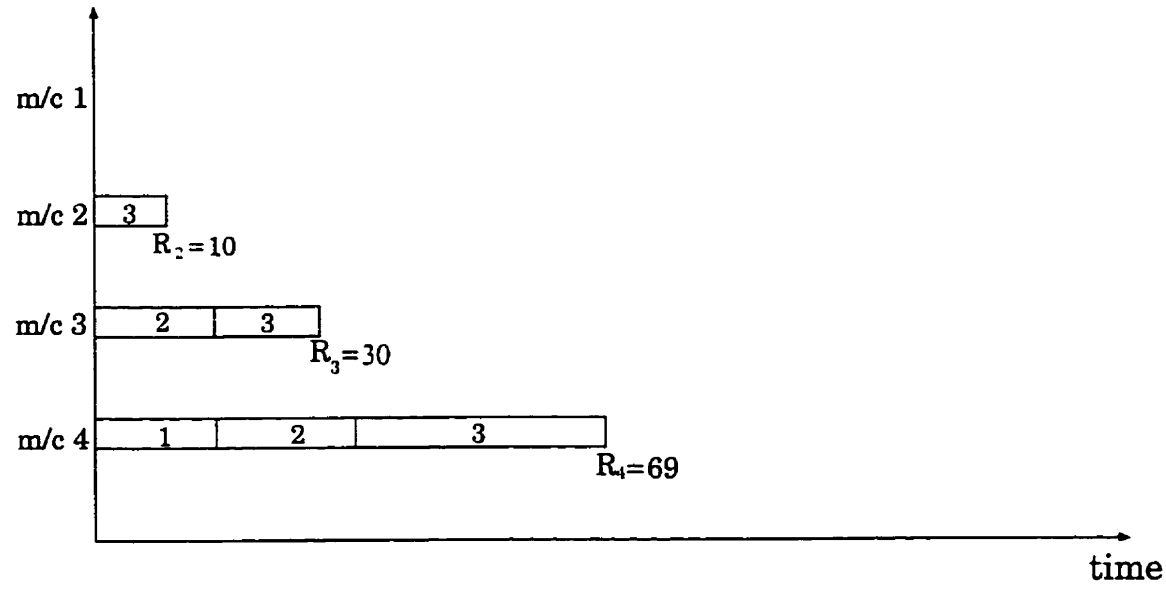
Table 3.2: Updated data for Example 3-1.

	MACHINE						
JOB	1	2	3	4	d_i	h_i	t_i
4	33	18	21	6	40	0	1
5	15	3	3	1	18	0	1

Example 3-1 illustrated a typical setting in which CD is invoked. Namely, a machine is available for dispatching at an instant in time, and sequence costs for



(a)



(b)

Figure 3.2: Gantt charts showing partial schedules.

all the machines have to be calculated considering the instantaneous status of each machine. The sequence cost, SC_{ky}, for machine k ($k > s$), given that the y^{th} job in Γ_s is selected for dispatching, is determined from three time-related components.

They are :

(a) Machine k Ready Time (R_k)

This variable represents the earliest time, measured from the present, at which machine k is able to start processing job x. R_k is calculated for each machine by assuming that all of the in-process jobs are processed on a first-come first-served (FIFO) basis. This is equivalent to simulating the current state of the system to completion by using FIFO dispatching, while withholding unstarted jobs from the cell. The completion time of the last job at each machine, as determined from this simulation, corresponds to that machine's ready time, R_k. Hence, R_k is effectively the time required for machine k to process, in order of arrival, all of its in-process workload. This workload includes all the current jobs in the cell that still have to pass through machine k. The ready times for stage 4 in Example 3-1 are shown in Figure 3.2(b). Clearly, given that the order of processing is defined in the buffers, the ready times at any of the machines can be calculated straightforwardly.

(b) Earliest finishing time for the dispatched job x on machine k ($\lambda_{x,k}$)

Once job x is dispatched on machine s, it will reach machine k ($k > s$) on its route at a future instant. $\lambda_{x,k}$ represents the earliest time that job x can be completed on machine k, assuming that the servicing of all the in-process jobs on the machines between s and k is done in the order of FIFO. Determin-

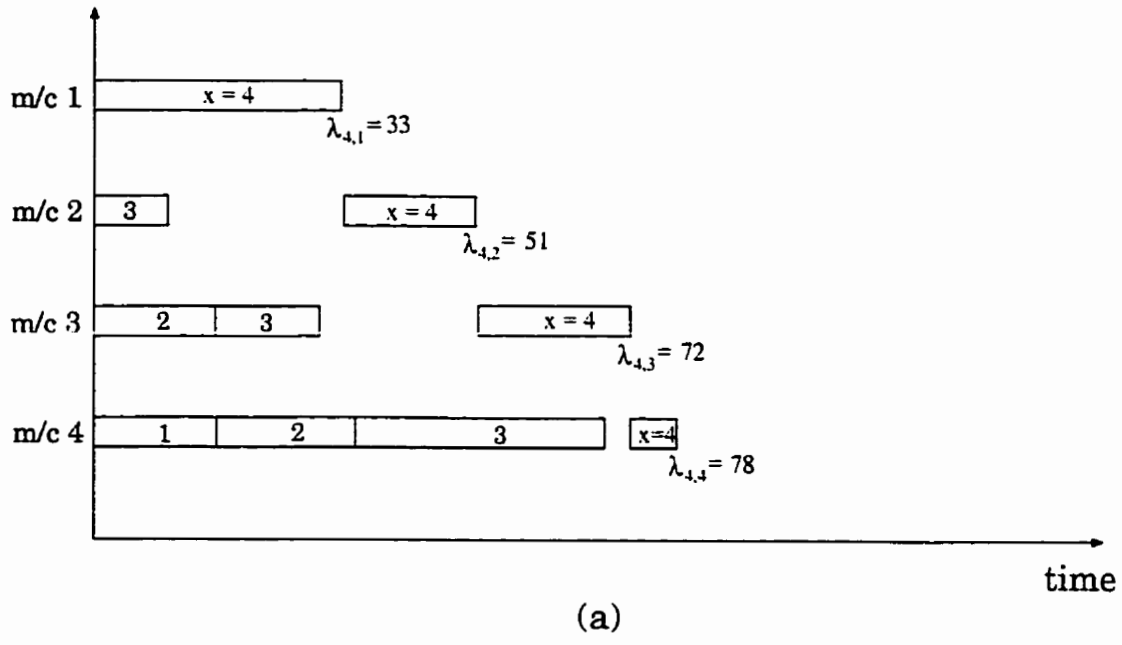
ing $\lambda_{x,k}$ requires prior knowledge of R_k. Therefore, the determination of R_k is prerequisite for calculating $\lambda_{x,k}$. Figure 3.2(b) illustrates how R_k appears after the third stage in Example 3-1. Figure 3.3 (a) displays $\lambda_{x,k}$ if x is job 4 and Figure 3.3 (b) shows the situation in the event that job 5 is the one chosen for dispatching on machine 1.

(c) Minimum cost for processing the remaining jobs on machine k.

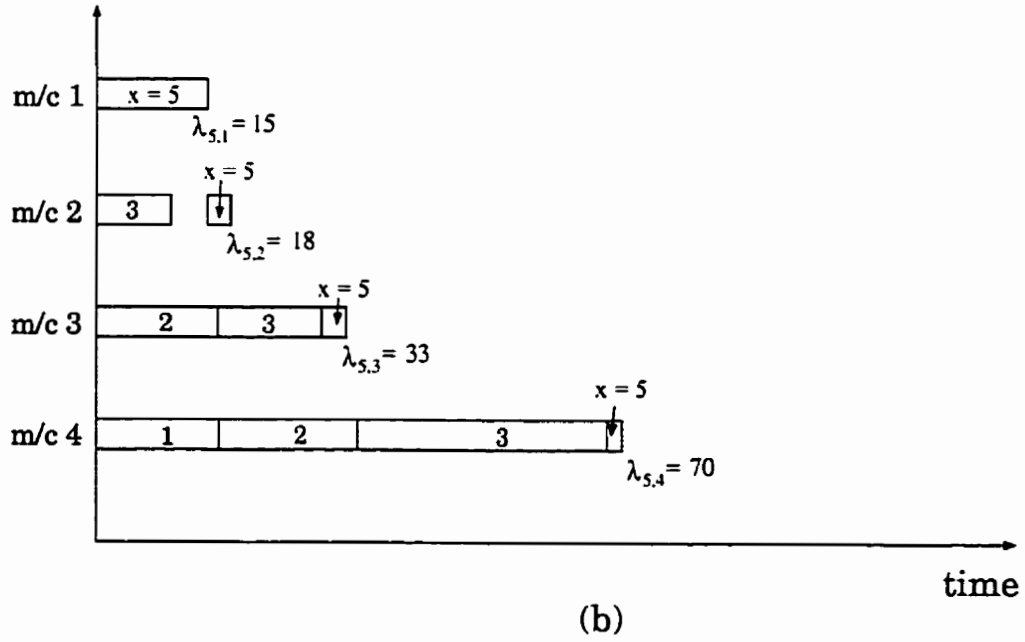
If job x is selected for dispatching, the earliest time that the set of remaining jobs (excluding job x) can start on machine k is at $\lambda_{x,k}$. (See Figures 3.3 (a) and (b).) The remaining jobs can be sequenced in a manner which optimizes the performance measure (i.e. cost minimization) strictly on machine k alone. Thus the cost calculated is a lower bound because all the jobs in Ω_k (with the exception of job x) are assumed to be available immediately (i.e. their ready times are ignored). This component of SC_{kx} is a single machine sequencing problem. The objective is to optimally sequence the jobs in Ω_k, given that job x is first in the sequence. The **core sequence** for machine k is a term that defines a sequence of jobs in Ω_k, starting at $\lambda_{x,k}$ and excluding job x.

3.2.2.1 Calculation of machine ready time (R_k)

The machine ready time, R_k, at a given instant for each machine k in the cell is calculated by using Algorithm S-1 and the known current status of all the cell machines and buffers. This algorithm is a subroutine that is called by the main algorithm whenever information concerning the future availability of the machines



(a)



(b)

Figure 3.3: Computing $\lambda_{x,k}$ for (a) $x = 4$ and (b) $x = 5$ from Example 3-1.

is required. Algorithm S-1 merely simulates the evolution of the in-process flow in the cell up to a point that allows the determination of when, at the earliest, each machine will be able to accommodate a newly dispatched part. This simulation merely tracks the event instances leading to this point. An event occurs whenever a machine completes its current job and becomes available for the next. Algorithm S-1 is described next.

Algorithm S-1

Let

t = time.

s = machine where dispatching is required.

k = machine number $k = 1, 2 \cdots, m$.

Ψ_k = set of jobs lined up at the buffer for machine k.

a_k = job currently in-process at machine k.

u_k = time taken to complete current operation on machine k.

η = most recently dispatched job in the cell.

ϕ = triplet (a,b,c) that describes an event, where

a = time of the event's occurrence.

b = job involved in the event.

c = machine where the event occurs.

E = set of the active triplets having the form ϕ.

Step 1. **Initialization.**

$E = \emptyset$.

$R_k = 0 \; \forall \; k$.

For all k, if $a_k \neq 0$, then $\phi = (u_{a_k}, a_k, k)$ is appended to E.

Step 2. **Identification of Event.**

If $E = \emptyset$, then continue to Step 6.

Otherwise determine $\phi \in E$ which has the earliest event occurrence amongst the triplets current in E. Let σ^* be the time of this occurrence, m^* be the machine concerned, and j^* be the job for this event.

Step 3. **Job Transfer to Next Machine.**

If m^* is not the last machine in the route for job j^*, remove j^* from a_{m^*} and add it to Ψ_k where k is the next machine in the route of j^*.

If $a_k = 0$, then $a_k = j^*$ and $\Psi_k = \emptyset$.

Step 4. **Loading Current Machine.**

Remove $\phi = (\sigma^*, j^*, m^*)$ from E.

If $m^* = \eta$, then $R_{m^*} = t$.

Step 5. **Updating Event List.**

If $\Psi_{m^*} \neq 0$, then

a_{m^*} = next job waiting in line at the buffer for machine m^*; and

$u_{a_{m^*}}$ = processing time for job a_{m^*} on machine m^*.

Add $\phi = (\sigma^* + u_{a_{m^*}}, a_{m^*}, m^*)$ to E.

Return to Step 2.

Step 6. **Termination.**

Return values calculated for R_k, $k = 1, 2, \cdots, m$ to the calling Algorithm.

Stop.

Algorithm S-1 is iterative. An iteration starts with the completion of a job on one of the machines (step 2), and covers the transfer of this job to its next destination (step 3), the loading of the now vacant machine with the next part in its buffer (step 4), and ends with the updating of the cell's status (step 5).

Example 3-2

To demonstrate the application of Algorithm S-1, the machine ready times (R_k) are calculated at the beginning of stage 4 for Example 3-1. The instantaneous situation, with machine 1 waiting to be loaded, is illustrated in the Gantt chart of Figure 3.2 (b). Step 1 initializes the variables to reflect the current status of the cell at instant t=0. At the start of stage 4, jobs 3, 2 and 1 are in-process on machines 2, 3 and 4 respectively, and machine 1 is available for loading. Thus, $a_1 = 0$; $a_2 = 3$; $a_3 = 2$; and a_4 =1. The times remaining to complete the current jobs on machines 2, 3 and 4 are 10, 16 and 16 units, respectively, so that $u_1 = 0$, $u_2 = 10$, $u_3 = 16$ and $u_4 = 16$. R_1 through R_4 are initialized to zero, and Ψ_1 to Ψ_4 are empty because there are no jobs waiting in the intermediate buffers. The events list, E, at time t=0 is {(10,3,2),(16,2,3),(16,1,4)}. The job dispatched in the previous stage, η, is job 3 (i.e. $\eta = 3$).

Iteration 1

Job 3 is completed on machine 2 at time t=10. Then it is unloaded and moved to the buffer for machine 3. As job 3 is the job identified by η, its completion signals that the machine ready time for machine 2 is equal to t . Therefore $R_2 = 10$. E is now updated and appears as {(16,2,3),(16,1,4)}.

Iteration 2

E shows that job 2 on machine 3 and job 1 on machine 4 are completed simultaneously. This tie is settled arbitrarily, and job 2 is unloaded from machine 3 and moved to the buffer for machine 4. Job 3, waiting in the buffer, is loaded next on machine 3. E now becomes {(16,1,4),(30,3,3)}.

Iteration 3

Job 1 is completed at time t=16 and it is unloaded from machine 4. Job 2 is loaded next on machine 4, and E is now {(30,3,3),(35,2,4)}.

Iteration 4

Job 3 is completed on machine 3 at time t=30. It is unloaded and moved to the buffer for machine 4. R_3 equals 30 because the job just unloaded was the job η. E is now {(35,2,4)}.

Iteration 5

Job 2 is completed and unloaded from machine 4 at time t =35. Job 3, waiting in the buffer for machine 4, is then loaded. E is {(69,3,4)}.

Iteration 6

Job 3 is completed and unloaded from machine 4 at time t=69. Therefore, R_4 is equal to 69 ($\eta = 3$). E = $\emptyset$ now, and the Algorithm S-1 terminates at this point.

Algorithm S-1 returns the following machine ready times for Example 3-1.

$R_1 = 0$; $R_2 = 10$; $R_3 = 30$; and $R_4 = 69$.

These results can be verified with reference to Fig. 3.2(b).

3.2.2.2 Calculation of $\lambda_{x,k}$

The earliest time that the dispatch candidate (job x) can be completed on machine k, $\lambda_{x,k}$, depends on whether or not machine k belongs to the set of remaining machines that will be visited by job x (β_x). If machine k is visited by job x ($k \in \beta_x$), then $\lambda_{x,k}$ is influenced directly by the earliest finishing times for job x on the preceding machines on the route of job x. If job x does not visit machine k ($k \notin \beta_x$), then $\lambda_{x,k}$ does not theoretically exist. Nevertheless, the processing of job x on machines preceding k in the flowshop indirectly influences the earliest completion times for other jobs that share some machines with job x and that do visit machine k. Consequently, an artificial job is created for use in determining a value for $\lambda_{x,k}$ in the case where machine k is not visited by job x.

Case I : job x visits machine k ($k \in \mathcal{J}_x$)

An example of such a situation would occur in the Type III configuration depicted in Figure 3.1. Supposing that job x's route is $\mathcal{J}_x = \{1,2,4\}$, then $\lambda_{x,k}$ would be calculated when k is 1, 2, or 4 under this case. Here, job x can start on machine k if and only if this machine is available, and the operation for job x on the preceding machine in job $x's$ route is completed, as illustrated in Figure 3.4.

With the exception of the pure flowshop (Type I) configuration, a job need not always go next to the machine having the immediately following index number. Therefore, the index μ_k identifies the index number of machine k in the set $\mathcal{J}_x$. Thus machine k will be the μ_k^{th} machine in the remainder of the route for job x.

$$\mu_k = \arg_{1 \le j \le n_{\mathcal{J}_x}} (\mathcal{J}_x^j = k). \tag{3.2}$$

To illustrate, if $\mathcal{J}_x = \{1,2,4\}$, then Equation (3.2) gives μ_1=1; μ_2=2; and μ_4=3; μ_3 does not exist (because $3 \notin \mathcal{J}_x$).

Referring to Figure 3.4, it is seen that $\lambda_{x,k}$ is equal to the maximum of the earliest starting time on machine k (R_k) and the earliest finishing time for job x on the machine preceding k on the route of job x ($\beta_x^{\mu_k - 1}$), plus the processing time for job x on machine k. Thus, $\lambda_{x,k}$ can be calculated by :

$$\lambda_{x,k} = \max\{R_k, \lambda_{x,\beta_x^{\mu_k-1}}\} + p_{x,k}, \qquad k \in \mathcal{J}_x \tag{3.3}$$

$$\lambda_{x,0}, \beta_x^0 = 0.$$

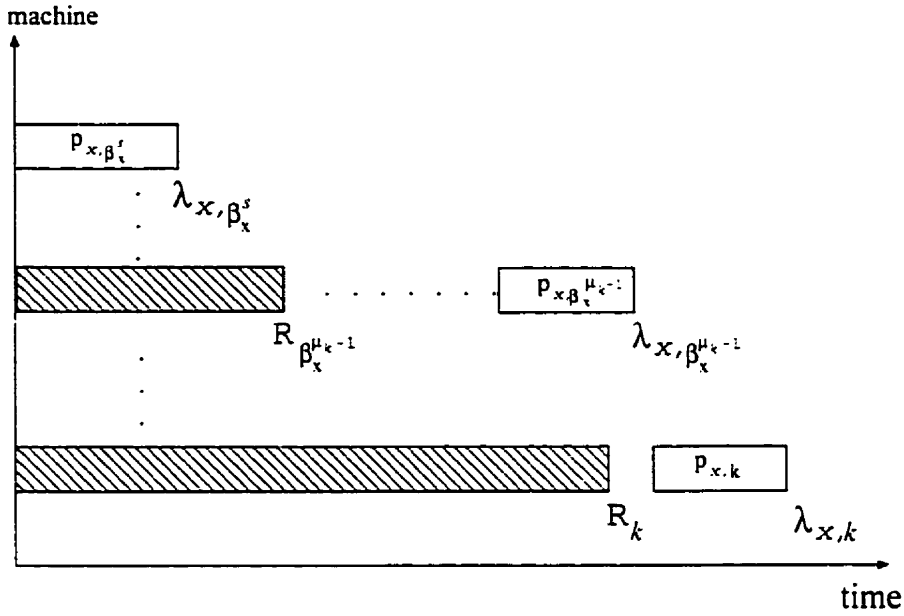


Figure 3.4: $\lambda_{x,k}$ when $k \in \mathcal{J}_x$.

Case II : job x does not visit machine k ($k \notin \mathcal{J}_x$)

Consider the same example as that used in Case I. Here $\lambda_{x,k}$, when $k = 3$, would fall under a Case II situation because job x's route does not include machine 3. Application of Equation (3.3) in this instance would leave $\lambda_{x,k}$ equal to R_k. In reality, however, the selection of job x at machine s when $k \notin \mathcal{J}_x$ implies that other dispatching candidates that do pass through machine k will be delayed in reaching k, the more so as they share preceding machines with job x. Consequently, allowing $\lambda_{x,k} = R_k$ here would result in a substantial underestimation of the sequence cost in machine k. This would cause machine k to gain an unwarranted weight in the dispatching decisions. To find a closer estimate of $\lambda_{x,k}$, an 'artificial' job is created and used as a 'dummy' for job x. The processing time on machine k for this artificial job is taken simply as the average of the processing times of all the jobs in Ω_k that

also visit machine s. As job x precedes the artificial job in all machines in β_x between machine s and machine j, a set of λ_{x,β_x^j} for $s < j < k$ would exist, as shown in Figure 3.5. The machine having the maximum λ_{x,β_x^j} is identified by :

$$l(x) = \arg \max_{s \le j < k} (\lambda_{x,\beta_x^j}). \tag{3.4}$$

When $l(x)$ is found, then $\lambda_{x,k}$ may be estimated by adding the duration of the artificial job on machine $l(x)$ to $\lambda_{x,l(x)}$ as follows:

$$\lambda_{x,k} = \max\{R_k, \lambda_{x,l(x)}\} + \frac{1}{n_{\Omega_k}} \sum_{i \in \Omega_k} p_{i,l(x)}, \qquad k \notin \beta_x \tag{3.5}$$

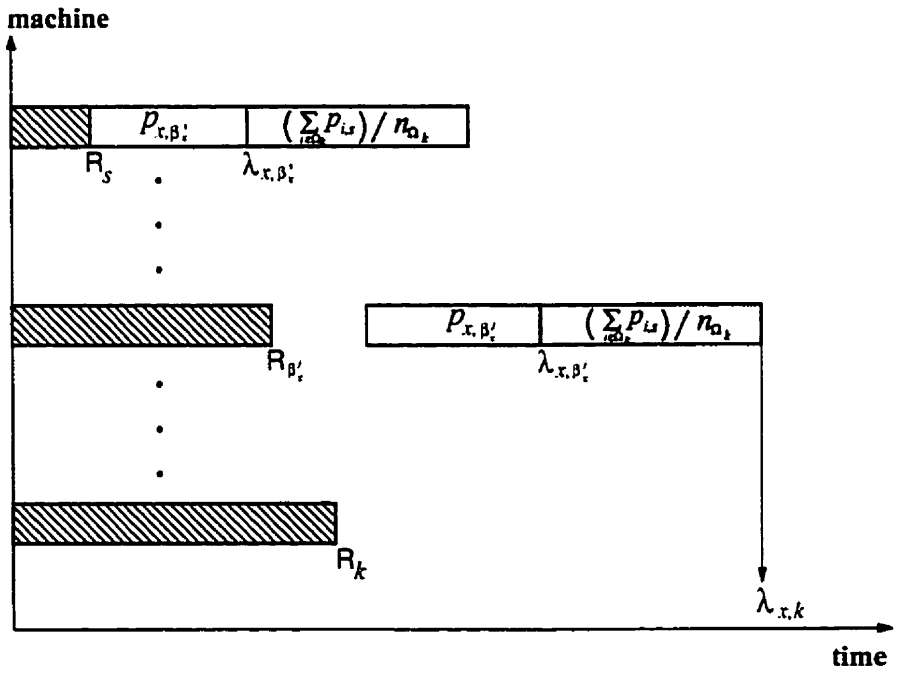


Figure 3.5: $\lambda_{x,k}$ when $k \notin \beta_x$.

Equation (3.5) is similar to Equation (3.3) except that machine $l(x)$ and the artificial job's processing time are considered rather than machine $\beta_x^{\mu_k - 1}$ and the processing

time of job x respectively. Equations (3.3) and (3.5) can be combined into one general equation to determine $\lambda_{x,k}$ as :

$$\lambda_{x,k} = \begin{cases} \max\{R_k, \lambda_{x,\beta_x^{(\mu_k-1)}}\} + p_{x,k} & \text{if } k \in \mathcal{J}_x \\ \max\{R_k, \lambda_{x,l(x)}\} + \frac{1}{n_{\Omega_k}} \sum_{i \in \Omega_k} p_{i,l(x)} & \text{if } k \notin \mathcal{J}_x \end{cases} \qquad (3.6)$$

$$\mathcal{J}_x^0, \lambda_{x,0} = 0$$

Equation (3.6) allows the computation of $\lambda_{x,k}$, in general, for all the configurations considered in Figure 3.1.

Example 3-3

The following example illustrates how $\lambda_{4,2}$ is found from Equation (3.6) in stage 4 of the problem described in Example 3-1. Equation (3.6) indicates that $\lambda_{x,k}$ is calculated recursively. This means that, to find $\lambda_{4,2}$, values of $\lambda_{4,1}$, and $\lambda_{4,0}$ must be available beforehand. Note that the ready times for the four machines (R_1=0,R_2=10,R_3=30,R_4=69) have been calculated already in Example 3-2, and that the route for job $x = 4$ is $\mathcal{J}_4 = \{1,2,3,4\}$. Hence, by applying Equation (3.3), we have $\lambda_{4,0} = 0$.

To find $\lambda_{4,1}$, Equation (3.2) is used to find μ_1 :

$$\mu_1 = arg_{1 \le j \le 4}(\beta_4^j = 1) \Rightarrow \mu_1 = 1.$$

Applying Equation (3.6), and noting that β_4^0=0;

$$\lambda_{4,1} = \max\{R_1, \lambda_{4,0}\} + p_{4,1} = 33.$$

To find $\lambda_{4,2}$, from Equation (3.2)

$$\mu_2 = arg_{1 \leq j \leq 4}(\beta_4^j = 2) \Rightarrow \mu_2 = 2.$$

From Equation (3.6), and noting that β_4^1=1;

$$\lambda_{4,2} = \max\{R_2, \lambda_{4,1}\} + p_{4,2} = 51.$$

This result may be verified by using the Gantt chart presented in Figure 3.3 (a).

3.2.2.3 Determining the Core Sequence for Machine k

Once $\lambda_{x,k}$ is known, the jobs remaining in machine k's workload (Ω_k) can be sequenced as a single machine problem starting at time $\lambda_{x,k}$. Thus $\lambda_{x,k}$ is the instant in time after which the assumption of single machine sequencing holds. The order in which the jobs in Ω_k are sequenced influences the performance measure when it is applied to machine k alone. A sequence's effectiveness in meeting the performance measure may be expressed as a cost function in the form of :

$$f_{k,y}(S) = \min \sum_{\substack{i \in \Omega_k \\ i \neq x}} [(\lambda_{x,k} + C_i(S))h_i + \delta(\lambda_{x,k} + C_i(S) - \Delta_{i,k})t_i] \qquad (3.7)$$

where $\delta(f) = \begin{cases} f & \text{if } f > 0 \\ 0 & \text{if } f \leq 0 \end{cases}$

and $k = 1, 2, \cdots, m$; $\quad y = 1, 2, \cdots, n_{\Gamma_s}$.

There is a sequence (S^*) of jobs i in Ω_k that will minimize $f_{ky}(S)$ in Equation (3.7). This sequence is the desired one because it will reflect, by minimizing $f_{k,y}(S)$, machine k's best ability to satisfy the performance measure. Thus,

$$f_{k,y}(S^*) = \min f_{k,y}(S). \tag{3.8}$$

The last component of SC_{ky} is derived, therefore, after finding an optimal solution for the core sequence (i.e $f_{k,y}(S^*)$). The difficulty of solving this single machine problem depends on the performance measure that is in force. When minimizing the mean flowtime, for instance, SPT sequencing of the jobs in the core will produce the optimal solution (and, hence, the lowest cost for SC_{ky}). The optimal solution for other performance measures cannot always be obtained by simple sorting, and implicit enumeration (i.e. mathematical programming) may be required. An example of such a case is minimizing the mean tardiness. The core sequence may be optimized for this particular objective by using dynamic programming.

Finding a core sequence that minimizes the mean tardiness on machine k requires due dates for job completions on machine k. If a job's final due date, d_i, is used, there will exist an increased possibility of machine indifference in the core sequence on account of too much slack in the due dates. This is because d_i is based on the job passing through multiple machines in the cell, and not just one. A remedy is to apply operation-based due dates. The operation due date is the time by which the processing of job i on machine k must be completed. There are several methods for assigning operation due dates. The one adopted here uses the latest time that

the processing of job i can be completed on machine k without contributing to the tardiness in completing job j. The operation due dates are calculated as :

$$\Delta_{i,k} = \begin{cases} \Delta_{i,(k+1)} - p_{i,(k+1)} & k = 1, 2, \cdots, m \\ d_i & k = m. \end{cases} \tag{3.9}$$

3.2.2.4 Dynamic Programming Formulation

The performance measure described by Equation (3.7) is a function of the completion times of the jobs. A sequence that minimizes function (3.7) may be found by using dynamic programming (DP). The DP method used here is based on [65]. and it is described next.

Let $Q_i = \lambda_{x,k} + C_i$. Then function (3.7) can be rewritten as :

$$min \ Z = \sum_{i=1}^{n} g_i(Q_i) \tag{3.10}$$

where

$$\begin{aligned} g_i(Q_i) &= t_i(Q_i - \Delta_{ik}) + h_i Q_i \qquad \text{if } Q_i > \Delta_{ik} \\ &= h_i Q_i \qquad\qquad\qquad\quad \text{if } Q_i \leq \Delta_{ik}. \end{aligned}$$

Suppose

K = set of all jobs to be scheduled.

J = set of all the unscheduled jobs in K.

$G(J)$ = minimum cost for the jobs in J given that $(K - J)$ jobs have been scheduled.

The recursive equation used in this dynamic programming formulation is :

$$G(J) = min\ [g_i(Q_i) + G(J - j)] \tag{3.11}$$

where $$G(0) = 0. \qquad j \in J.$$

The solution from the dynamic program results in a set of jobs that is ordered in the sequence that minimizes the mean tardiness.

Having examined the three components of the sequencing cost and the details of their calculations, it is now possible to describe the equation that is used in calculating the sequence costs.

3.2.2.5 Calculating SC_{ky}

SC_{ky}, the cost of an optimal sequence on machine k, is calculated under the condition that a specific job, $x \in \Gamma_s$, is the first in that sequence. This cost is weighted to reflect the global importance of the machine in the cell. The weighted values of SC_{ky} are computed by multiplying the sequence costs for each machine k by a factor, W_k. There are several ways to quantify the 'importance' (W_k) of

machine k. The one used here is based on the instantaneously remaining workload at each machine. The higher is the remaining work on one machine, when compared to that for another machine, the greater is its influence in the dispatching decision. The following equation describes how the weights are defined for each machine:

$$W_k = L_k + \sum_{i \in \Gamma_k} p_{i,k} + \sum_{i \in \Omega_k} p_{i,k}. \tag{3.12}$$

where L_k is the process time of the job currently occupying machine k.

Letting

$$W_{max} = \max\{W_k\} \qquad k = 1, 2, \cdots, m. \tag{3.13}$$

Then the SC matrix is constructed by using :

$$SC_{ky} = \frac{W_k}{W_{max}} \times [h_{\Gamma_s^y} \lambda_{\Gamma_s^y,k} + \delta(\lambda_{\Gamma_s^y,k} - \Delta_{\Gamma_s^y}) t_{\Gamma_s^y} + f_{ky}(S^*)]. \tag{3.14}$$

Example 3-4

Reconsider the problem given in Table 3.1 for Example 3-1. SC_{21} will be calculated next for stage 4 in the solution to that problem. Here $\Gamma_s = \{4,5\}$ and $\Omega_{k=2}$ is $\{4,5\}$. (See Figure 3.3 (a).) SC_{21} refers to the sequence cost on machine 2 if the first job (y=1) in the buffer for machine s is dispatched next. Hence $\Gamma_s^y = \Gamma_1^1$ = 4. The set of remaining jobs available for the core sequence is $S = \{5\}$. The optimal solution for the core sequence is trivial because there is only one job in S.

Hence, S^*={5}. Had there been more than one job in S, then S^* would have been a sequence of those jobs that minimizes the mean tardiness.

Noting that $\Gamma_1^1 = 4$ and using Equation (3.14) gives:

$$SC_{21} = \frac{W_2}{W_{max}}[h_4\lambda_{4,2} + \delta(\lambda_{4,2} - \Delta_{4,2})t_4 + f_{21}(\{5\})] \ .$$

From Equations (3.12) and (3.13):

$W_1 = 33 + 15 = 48;$

$W_2 = 10 + 18 + 3 = 31;$

$W_3 = 27 + 14 + 21 + 3 = 65;$

$W_4 = 27 + 19 + 34 + 6 + 1 = 87.$

$$W_{max} = \max\{48.31.65.87\}.$$

From Equation (3.9), the operational due date for job 5 on machine 2 is:

$$\Delta_{5,2} = 18 - (1 + 3) = 14.$$

A value for $f_{21}(S^*)$ is needed in calculating SC_{ky}. It is found by using Equations (3.7) and (3.8) as follows :

$$f_{21}(\{5\}) = [\lambda_{\Gamma_1^1,2} + C_5(\{5\})] \times h_5 + \delta(\lambda_{\Gamma_1^1,2} + C_5(\{5\}) - \Delta_{5,2}) \times t_5.$$

Since $\Gamma_1^1 = 4$;

$$f_{21}(\{5\}) = [\lambda_{4,2} + C_5(\{5\})] \times h_5 + \delta(\lambda_{4,2} + C_5(\{5\}) - \Delta_{5,2}) \times t_5.$$

Recalling from Example 3-2 that $\lambda_{4,2} = 51$ for this situation:

$$f_{21}(\{5\}) = [51 + 3] \times 0 + \delta(51 + 3 - 14) \times 1.0. = 40.$$

Thus, by using Equation (3.14) to compute SC_{21}:

$$SC_{21} = \frac{31}{87}[h_4\lambda_{4,2} + \delta(\lambda_{4,2} - \Delta_{4,2})t_4 + f_{21}(\{5\})] \ .$$

From Equation (3.9):

$$\Delta_{4,2} = 40 - (21 + 6) = 13$$

so that,

$$SC_{21} = 0.36 \times [0 + \delta(51 - 13) + 40] = 27.79 \ .$$

Similar calculations produce $f_{11} = 37$; $f_{12} = 53$; $f_{22} = 23$; $f_{31} = 58$; $f_{32} = 20$; $f_{41} = 61$; and $f_{42} = 36$.

The use of Equation (3.14) results in the following SC matrix for stage 4 of Example 3-1:

$$SC = \begin{bmatrix} 41.38 & 31.45 \\ 27.79 & 9.62 \\ 71.72 & 26.90 \\ 99.00 & 88.00 \end{bmatrix}.$$

3.2.3 Selection of Job for Dispatching

Once the SC matrix is complete, it becomes possible to evaluate the consequences of various dispatching decisions. In particular, the negative impact on the machines whose candidate jobs are not selected can be assessed by measuring the resulting increments in the sequence costs. These increments can be viewed as penalties. For example, if machine k's candidate is job x, but job x' is the one that is dispatched, then the penalty suffered by machine k will be $SC_{ky'} - SC_{ky}$, where y and y' are the positions in the queue of the jobs numbered x and x' respectively.

The 'cooperative' nature of the above job selection technique results from the fact that penalties are assigned from the perspective of the machine rather than the job. In other methods where job dispatching is based on minimum penalty schemes, such as Gelders and Sambandam [47] or Das et. al. [66], an index is computed in order to rank each job's augmentation to a partial schedule. This index is based on the job's contribution to the performance criterion for the system as a whole.

In cooperative dispatching, conversely, the main entities are the machines, not the jobs, and the penalties are a measure of the deviation from the optimal satisfaction of the performance criterion locally, with respect to each machine. The final job dispatching decision reflects a resolution of these, usually conflicting, requirements for local optimization. The conflicts are resolved through a process that involves nominations of preferred jobs from each of the machines, followed by the selection of a winning candidate.

3.2.3.1 Identifying Candidates

The next step is to determine which of the jobs in Γ_s are preferred by each of the machines involved in the dispatching decision. Every machine, as explained previously, makes a first and a second choice, D'_k and D''_k respectively, as follows:

$$D'_k = arg \min_y SC_{ky} \qquad k = 1, 2, \cdots, m. \tag{3.15}$$

$$D''_k = arg \min_{y \neq D'_k} SC_{ky} \qquad k = 1, 2, \cdots, m. \tag{3.16}$$

and

$$D_k = D'_k \cup D''_k \qquad k = 1, 2, \cdots, m. \tag{3.17}$$

The set of jobs nominated for dispatch, labeled set G, is

$$G = D_1 \cup D_2 \cup \cdots D_k \qquad k = 1, 2, \cdots, m. \tag{3.18}$$

The candidate jobs in set G are referenced by their locations in the buffer for machine s. If $G = \{2,4,5\}$, for example, then the three candidates are the second, fourth and fifth jobs waiting in line at the buffer for machine s.

3.2.3.2 Determining the Winning Job.

The members of set G owe their priorities to different machines that have conflicting interests. The dispatching selection revolves around the need to compromise between those conflicting nominations. This compromise is done by calculating a cumulative penalty cost, ρ_y, that is incurred when job $y \in G$ is selected. This penalty is calculated from:

$$\rho_y = \sum_{k=1}^{m} (SC_{ky} - v_k) \quad \forall \ y \in G. \tag{3.19}$$

where v_k, the minimum sequence cost for machine k, is

$$v_k = \min_y SC_{ky} \qquad k = 1, 2, \cdots, m \tag{3.20}$$

The job winning the dispatching competition, designated job y^*, is the one producing the minimum ρ_y. Therefore,

$$y^* = \arg\min_{y \in G} \rho_y. \tag{3.21}$$

The job to be dispatched next is, therefore, the job occupying the y^{*th} position in queue at the buffer for machine s. This job is identified by $\Gamma_s^{y^*}$. An example is employed next to illustrate the procedure of deciding, by means of the SC matrix, which job, from a set of available jobs, is dispatched next.

Example 3-5

Suppose that, at a certain stage, job numbers 6, 3, 1, 4 and 8 are queued in that order in the buffer for machine s. Assume the corresponding sequence cost matrix is:

$$SC = \begin{bmatrix} 251.00 & 307.00 & 299.00 & 245.00 & 251.00 \\ 82.02 & 143.07 & 125.83 & 81.55 & 81.55 \\ 272.43 & 442.14 & 384.97 & 337.63 & 306.37 \\ 235.66 & 455.27 & 449.36 & 388.54 & 379.25 \end{bmatrix}.$$

The set of available jobs being considered for dispatching is $\Gamma_s = \{6,3,1,4,8\}$.

From Equation (3.20)

$v_1 = 245.00$; $v_2 = 81.55$; $v_3 = 272.43$; and $v_4 = 235.66$.

The candidate jobs are identified by using Equations (3.15) through (3.18) :

$D'_1 = 4$; $D'_2 = 4$; $D'_3 = 1$; and $D'_4 = 1$.

$D''_1 = 1$: $D''_2 = 1$: $D''_3 = 5$; and $D''_4 = 5$.

$D_1 = \{4.1\}$: $D_2 = \{4.1\}$: $D_3 = \{1.5\}$: and $D_4 = \{1.5\}$.

and

$G = \{1.4.5\}$.

Consequently, the candidate jobs for dispatching are the first, fourth and fifth jobs in the buffer (i.e. job numbers 6, 4 and 8).

Equation (3.19) is employed now to determine the winning candidate. Then,

$$\rho_1 = (SC_{11} - v_1) + (SC_{21} - v_2) + (SC_{31} - v_3) + (SC_{41} - v_4)$$
$$= (251.00 - 245.00) + (82.02 - 81.55) + (272.43 - 272.43) + (235.66 - 235.66)$$
$$= 6.47$$

$$\rho_4 = (245.00 - 245.00) + (81.55 - 81.55) + (337.63 - 272.43) + (388.54 - 235.66)$$
$$= 218.08$$

and

$$\rho_5 = (251.00 - 245.00) + (81.55 - 81.55) + (306.37 - 272.43) + (379.25 - 235.66)$$
$$= 183.53 .$$

A comparison of ρ_1, ρ_4 and ρ_5 shows that ρ_1 has the least numerical value. Hence the winning job is the first one in the buffer, which corresponds to job number 6 (because $\Gamma_s^1 = 6$). Therefore, CD dispatching results in job 6 being dispatched next on machine s.

3.2.4 Cooperative Dispatching Algorithm

Steps in a construction algorithm to implement the cooperative dispatching model are outlined next.

Step 1. Initialization.

1.1 $z_{kc} = 1$ if there is a job route from machine k to machine c. Otherwise, $z_{kc} = 0$ for $k = 1, 2, \cdots, m$ and $c = 1, 2, \cdots, m$.

1.2 Determine the operation due dates, Δ_{ik}, from Equation (3.9) for all jobs.

1.3 Set $CF_k = 0$ if machine k is constrained to FIFO dispatching only. Otherwise $CF_k = 1$.

1.4 $\xi = \emptyset$. $R_k = 0$ for $k = 1, 2, \cdots, m$.

1.5 If Γ_k^1 exists, then append (0.0,k) to ξ $\forall$ k.

1.6 Set $t = 0$.

Step 2. **Event Response.**

2.1 If ξ is empty, proceed to Step 6.

2.2 Find the $\phi \in \xi$ which has the earliest event time. Let s and α be the machine and job for this event and t^* be the time of its occurrence.

2.3 If $\alpha = 0$, put $x^* = 0$ and proceed to Step 5.

2.4 Set $t = t^*$. Subtract t^* from all Δ_{ik} and from all time elements, ϕ, in ξ.

2.5 The next machine on job α's route is $\mathcal{J}_\alpha^{\mu_s+1}$, where $\mu_s = \arg_{1 \le j \le n_{\mathcal{J}_\alpha}} (\mathcal{J}_\alpha^j = k)$. Let $k' = \mathcal{J}_\alpha^{\mu_s+1}$. If k' does not exist, then $\tau_\alpha = t^*$ and proceed to Step 3.

2.6 If machine k' exists and it is free, then job α is loaded on this machine and the corresponding event's triplet (ϕ) is added to the event list. Otherwise, job α is added at the end of the buffer for machine k'. Mathematically, if $U_k' = 0$, then $U_{k'} = \alpha$ and the event triplet $(p_{\alpha,k'}).\alpha,k'$ is appended to ξ. Otherwise, α is put at the end of the set $\Gamma_{k'}$.

2.7 If $CF_s \neq 1$, set $x^* = \Gamma_s^1$ and proceed to Step 5.

Step 3. **Construction of SC Matrix.**

3.1 If Γ_s^2 does not exist, set $x^* = \Gamma_s^1$ and proceed to Step 5.

3.2 SC_{ky} = -1.00 for $k = 1, 2, \cdots, m$; $y = 1, 2, \cdots, n_{\Gamma_s}$.
Determine Φ_s and set $x = 1$.

3.3 If Γ_s^x does not exist, proceed to Step 5.
Determine β_x.

3.4 Calculate $R_k \; \forall \; k \in \Phi_s$ by using sub-algorithm S-1.

3.5 Determine $\Omega_k \; \forall \; k \in \Phi_s$.

3.6 Calculate $SC_{ky} \; \forall \; k \in \Phi_s$ by using Equation (3.14).

3.7 Increment x by 1 and return to step 3.3.

Step 4. **Cooperative Dispatching Decision.**

4.1 Derive set G according to Equations (3.17) and (3.18).

4.2 Find $\rho_y \; \forall \; y \in G$.

4.3 Let x^* be the job having the minimum ρ_y.

4.4 Select job $\Gamma_s^{x^*}$ for dispatching.

Step 5. **Updates.**

5.1 Load job x^* on machine s and remove this job from Γ_s.

5.2 Add the event triplet ($p_{x^*,s}$, x^*, s) to ξ.

5.3 Return to Step 2.

Step 6. Termination

6.1 Calculate the performance measures by using the job completion times in the set τ.

6.2 Stop.

3.3 Numerical Example

The CD algorithm is used now to find a solution for a problem exemplifying a Type III configuration. The Type III configuration is selected simply to illustrate the CD algorithm for a flowshop configuration other than the pure flowshop case.

Example 3-6

Table 3.3 presents the processing times and performance costs for a four-job, four-machine sequencing problem. The performance criterion is the minimization of the mean flowtime. A zero processing time for a job on a given machine indicates that the job's route does not include that particular machine. Dispatching occurs for this problem only at machine 1, where the jobs are loaded into the cell. It is also given that the jobs in the intermediate buffers for machines 2, 3, and 4 are processed on a first-come, first-served (FIFO) basis. Thus, all the machines except machine 1 are constrained to a FIFO policy for servicing queues.

The solution found by CD for the problem of Table 3.3 is described next in a stage by stage fashion. The number of stages is three because only three dispatching

Table 3.3: Processing times for Example 3-6.

	MACHINE					
JOB	1	2	3	4	h_i	t_i
1	27	0	9	31	1	0
2	16	18	0	7	1	0
3	14	0	4	8	1	0
4	33	32	0	28	1	0

decisions are required at machine 1 in this problem. After the third stage only one job remains to be scheduled, and no selection decision is needed for that.

Initialization

The initialization step for this problem first defines the configuration. For the Type III configuration, $z_{01} = z_{12} = z_{13} = z_{24} = z_{34} = 1$. As machines 2, 3 and 4 are constrained to FIFO processing, then $CF_2 = CF_3 = CF_4 = 0$, and CF_1=1. Operational due dates are not initialized because the due dates play no role when the performance criterion is the minimization of the mean flowtime. The starting status in the cell is that jobs 1, 2, 3 and 4 are waiting, in a random order, for processing on machine 1. The randomized order used here gives $\Gamma_1 = \{2,1,4,3\}$. The buffers for the other machines are empty so that $\Gamma_2 = \Gamma_3 = \Gamma_4 = \emptyset$. The initial event list is $\xi = \{(0,0.1)\}$ and the current time is $t = 0$.

Table 3.4: Stage 1 calculations for Example 3-6.

s	(R_1, R_2, R_3, R_4)	y	k	Γ_s	β_x	Ω_k	x	$\lambda_{x,k}$	W_k	SC_{ky}
1	(0,0,0,0)	1	1	{2,1,4,3}	{1,2,4}	{2,1,4,3}	2	16	1.00	193.00
			2			{2,4}	2	34	0.56	55.56
			3			{1,3}	2	36	0.15	12.86
			4			{2,4,1,3}	2	41	0.82	226.11
1	(0,0,0,0)	2	1	{2,1,4,3}	{1,3,4}	{2,1,4,3}	1	27	1.00	215.00
			2			{2,4}	1	51	0.56	94.45
			3			{1,3}	1	36	0.15	10.98
			4			{2,4,1,3}	1	67	0.82	273.80
1	(0,0,0,0)	3	1	{2,1,4,3}	{1,2,4}	{2,1,4,3}	4	33	1.00	233.00
			2			{2,4}	4	65	0.56	82.22
			3			{1,3}	4	53	0.15	17.77
			4			{2,4,1,3}	4	93	0.82	361.78
1	(0,0,0,0)	4	1	{2,1,4,3}	{1,3,4}	{2,1,4,3}	3	14	1.00	191.00
			2			{2,4}	3	38	0.56	80.00
			3			{1,3}	3	18	0.15	6.5
			4			{2,4,1,3}	3	26	0.82	174.13

Stage 1

Machine 1 is available and a job from $\Gamma_1 = \{2,1,4,3\}$ is to be selected at time $t^* = 0$. The set of machines visited by the jobs in Γ_1 is $\Phi_1 = \{1,2,3,4\}$. Table 3.4 summarizes the current status of the variables that are used in Equation (3.14) to calculate the SC matrix shown in Table 3.5 for stage 1.

From the SC matrix at stage 1, G is found from step 4.1 to be $\{4,1,2\}$. Step 4.2

Table 3.5: SC matrix at stage 1.

	x			
k	2	1	4	3
1	193.00	215.00	233.00	191.00
2	55.56	94.45	82.22	80.0
3	12.86	10.98	17.77	6.5
4	226.11	273.80	361.78	174.13

gives $\rho_y = \{ 24.44, 60.16, 166.86 \}$, and $x^* = 4$ is determined in step 4.3. Thus, the fourth job waiting in the buffer for machine 1 (Γ_1^4) is selected. This is job number 3. Stage 1 ends, therefore, with the dispatch of job number 3 on machine 1. After that, Γ_1 becomes $\{1,2,4\}$ and the triplet (14,3,1) is added to ξ. ξ is now $\{(14,3,1)\}$.

Stage 2

Job 3 is finished on machine 1 at time $t^* = 14$, triggering stage 2. The triplet (14,3,1) is now removed from ξ, and t is reset to zero. All due dates and time data in ξ are reduced by 14 time units. The next machine on the route for job 3 is machine 3. Machine 3 is currently free, so job 3 is unloaded from machine 1 and loaded directly on machine 3. Triplet (4,3,3), which corresponds to the event marking the scheduled completion of job 3 on machine 3, is added to ξ. Machine 1 is to be loaded next by a job from $\Gamma_1 = \{1,2,4\}$. Table 3.6 shows the status at stage 2 for the variables that are used in finding the SC matrix, bearing in mind that Φ_1 is the set $\{1,2,3,4\}$. The SC matrix for stage 2 is presented in Table 3.7.

Table 3.6: Stage 2 calculations for Example 3-6.

s	(R_1, R_2, R_3, R_4)	y	k	Γ_s	β_x	Ω_k	x	$\lambda_{x,k}$	W_k	SC_{ky}
1	(0,0,4,12)	1	1	{2,1,4}	{1,2,4}	{2,1,4}	2	16	1.00	135.00
			2			{2,4}	2	34	0.66	65.79
			3			{1}	2	43	0.17	8.89
			4			{2,4,1}	2	41	0.97	204.48
1	(0,0,4,12)	2	1	{2,1,4}	{1,3,4}	{2,1,4}	1	27	1.00	146.00
			2			{2,4}	1	51	0.66	111.84
			3			{1}	1	36	0.17	6.16
			4			{2,4,1}	1	67	0.97	236.6
1	(0,0,4,12)	3	1	{2,1,4}	{1,2,4}	{2,1,4}	4	33	1.00	158.00
			2			{2,4}	4	65	0.66	97.37
			3			{1}	4	60	0.17	11.80
			4			{2,4,1}	4	93	0.97	315.47

G is found from the SC matrix at stage 2 to be {1,2,3}, and ρ_y is {2.74, 89.18, 171.22}. The smallest element in ρ_y is the first one, meaning that the first element in G is selected. This selection gives y^*=1, and the selected job is consequently Γ_1^1=2. Hence, job number 2 is dispatched to machine 1 during stage 2 of the CD algorithm. Γ_1 is now {1,4}, and the triplet (16,2,1) is added to ξ. ξ is consequently {(4,3,3),(16,2,1)}.

At time $t^* = 4$, job number 3 is finished on machine 3. The triplet (4,3,3) is removed from ξ and all the remaining triplets in ξ are updated by subtracting $t^* = 4$ time units from the event occurrence times. Thus, ξ at this point is {(12,2,1)}. The

Table 3.7: SC matrix at stage 2.

	x		
k	2	1	4
1	135.00	146.00	158.00
2	65.79	111.84	97.37
3	8.89	6.16	11.80
4	204.48	236.60	315.47

next machine on the route of job 3 is machine 4. Job 3 is unloaded off machine 3 and loaded directly on machine 4 (which is currently free). The triplet (8,3,4), which represents the completion time for job 3 on machine 4, is appended to ξ, so that ξ is now $\{(8,3,4),(12,2,1)\}$.

At time $t^* = 8$, job 3 is completed on machine 4 and unloaded out of the cell. The triplet for the job just completed is removed from ξ, and the remaining triplets are updated by subtracting $t^* = 8$ units so that ξ is now $\{(4,2,1)\}$.

Stage 3

At time $t^* = 4$, job number 2 is finished on machine 1. It is taken then from machine 1 and loaded directly on machine 2, which is the next machine on the route of job 2. The corresponding triplet (18,2,2) is put in ξ and it replaces the expired triplet (4,2,1). Machine 1 is now available, and this event marks the start of stage 3, where a job from the two remaining jobs ($\Gamma_1 = \{1,4\}$) is selected. Current values for the variables used in the computation of the SC matrix for stage 3 are given in Table 3.8, knowing that Φ_1 is $\{1,2,3,4\}$. The SC matrix is shown in Table 3.9.

Table 3.8: Stage 3 calculations for Example 3-6.

s	(R_1, R_2, R_3, R_4)	y	k	Γ_s	β_x	Ω_k	x	$\lambda_{x,k}$	W_k	SC_{ky}
1	(0,18,0,25)	1	1	{1,4}	{1,3,4}	{1,4}	1	27	0.91	79.09
			2			{4}	1	60	0.76	69.70
			3			{1}	1	36	0.14	4.91
			4			{4,1}	1	67	1.00	162.00
1	(0,18,0,25)	2	1	{1,4}	{1,2,4}	{1,4}	4	33	0.91	84.55
			2			{4}	4	65	0.76	49.24
			3			{1}	4	60	0.14	9.41
			4			{4,1}	4	93	1.00	217.00

The SC matrix of Table 3.9 is resolved in step 4 of the CD algorithm to give G= {1,2} and ρ_y = {0.44, 1.41}. The selected job is the one indicated by the first element in ρ_y (because it is numerically the least) and y^* = 1. From step 4.4, job Γ_1^1 = 1 is selected. Thus, job number 1 is dispatched to machine 1. The triplet (27,1,1) is put in ξ so that ξ is {(18,2,2),(27,1,1)} at the end of stage 3.

Table 3.9: SC matrix stage 3.

	x	
k	1	4
1	79.09	84.55
2	69.70	49.24
3	4.91	9.41
4	162.00	217.60

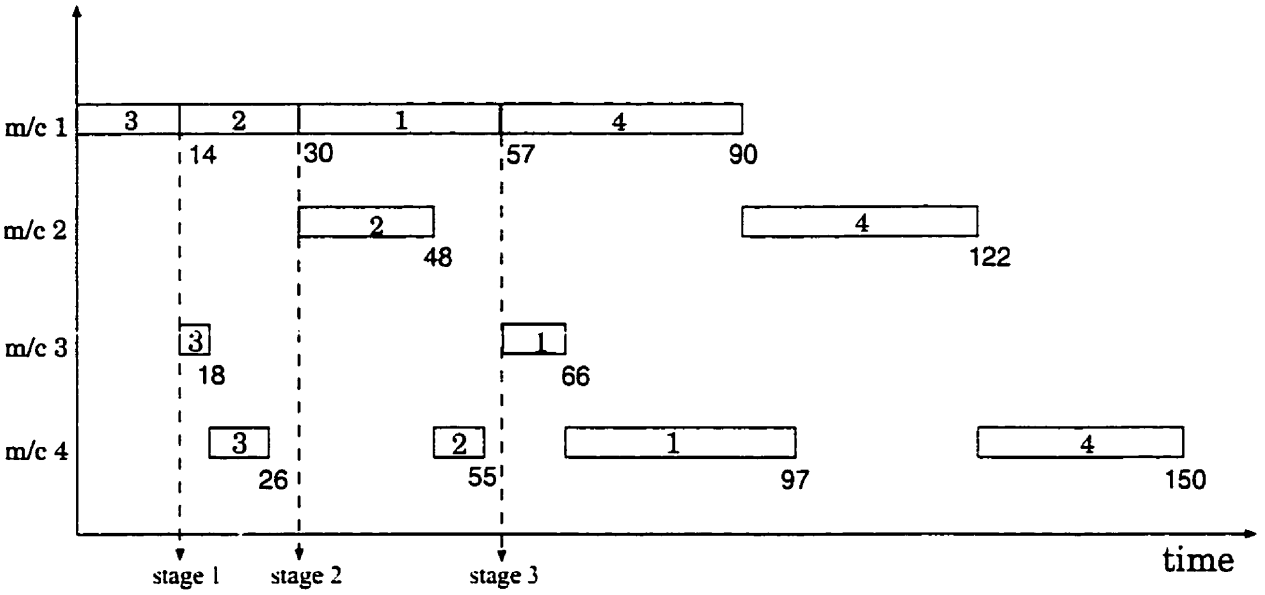


Figure 3.6: Gantt chart showing final schedule for the example problem.

CD dispatching has no further role to play in the solution because no more decisions are required. The algorithm continues by cycling between step 2 and step 5 as the remaining jobs are processed in FIFO order throughout the cell. The program terminates when all the jobs have exited the cell. The final solution provided by the CD algorithm in this example is to dispatch jobs at machine 1 in the sequence of jobs 3, 2, 1 and 4. The Gantt chart for this solution, which gives a mean flowtime of 82 units, is given in Figure 3.6.

3.4 Performance Evaluation

The performance of the CD algorithm is evaluated in this section in comparison with other procedures taken from the literature. The problems tested are strictly static sequencing problems. A static problem implies that the initial conditions correspond to, 1) no parts in-process in the cell; and 2) all the jobs are available and ready at the start of the scheduling period. Static cases provide the advantage of enabling scheduling to be performed off-line. Off-line scheduling allows a variety of methods to be used, including simulation, in order to search numerous sequences and to choose the best one. With on-line scheduling however, once a part is dispatched there is no chance to reverse that decision. The CD approach is a dispatching mechanism that is basically on-line oriented. It can also be used, nevertheless, as a single-pass procedure for the off-line, static sequencing of jobs.

3.4.1 Test Problems

The CD approach is evaluated by using sets of randomly generated test problems. The test problems are generated by using the same method as that described in Kim [46]. Each set contains 110 problems, all having a common number of jobs. A program written in the C computer language [75] and designed to simulate the activities in a FMC under the control of different dispatching rules is used in this comparative evaluation of CD. The simulation program processes an input data set according to the selected dispatching rule, and it calculates values for the corresponding performance measures. The abilities of CD and other dispatching rules to satisfy the given performance criteria are compared by using this program.

The computer simulations that are run in the evaluation of CD's performance cover cases involving the three routing configurations of Figure 3.1 and three different performance criteria. The evaluation is performed by comparing CD's performance with that of the dispatching rules that have been demonstrated in previous publications to be the most effective for the performance criterion under consideration. The results are expressed, whenever conveniently possible, in terms of the deviation from the optimal solutions. The optimal solution to each problem is found from an exhaustive enumeration of all the possible solutions.

Calculation of the optimal solutions, however, becomes prohibitive in problems with a large number of jobs due to the combinatorial explosion arising from complete enumeration. It is more convenient, then, to evaluate the CD algorithm relative to other dispatching procedures. The comparison is performed by taking as a reference a dispatching rule that is known to give acceptable results for the performance criterion under consideration, and then measuring CD's results against this reference. The measure that is adopted is called the performance ratio (PR), and it is defined by:

$$PR = m_r/m_a \tag{3.22}$$

where m_a and m_r are the results for the method under evaluation (i.e. CD) and the reference method respectively. A larger PR value above 1.0 indicates that the method being evaluated performs increasingly better than the reference method. A PR value near 1.0 signifies near-equivalent performance.

The tests are conducted for all three configurations shown in Figure 3.1. The Type I configuration is limited to three machines only. This is to provide a uni-

formity in total workload with the other two configurations, where the part routes pass strictly through three of the four machines that are available in each of those configurations.

3.4.2 Minimizing the Mean Flowtime

The performance of CD in minimizing the mean flowtime is evaluated by comparing its results with those obtained from using the Least Work Remaining (LWKR) dispatching rule. The LWKR rule is chosen for this purpose because it has been shown to be the better of several dispatching rules considered for flowshop configurations [46].

Figure 3.7 compares CD and LWKR in terms of percentage deviation from the optima. The test data includes six sets of problems, each set covering a problem size from 5 and upto 10 jobs. Figure 3.7 shows that the CD algorithm's results are within 5% of the optima. Furthermore, CD's performance appears to be quite consistent in all three configurations as the number of jobs grows to ten. On the other hand, the LWKR rule gives results generally deviating upwards of 5% from the optimal. Also, the quality of the results from the LWKR shows a pronounced deterioration with more jobs.

The performance of CD for problems having upto 50 jobs is presented in Figure 3.8. Figure 3.8 uses a performance ratio (PR) with LWKR as the reference method. The results indicate that, as the number of jobs grows beyond 15, the CD algorithm outperforms the LWKR dispatching rule by about 5 to 10%. The CD improves

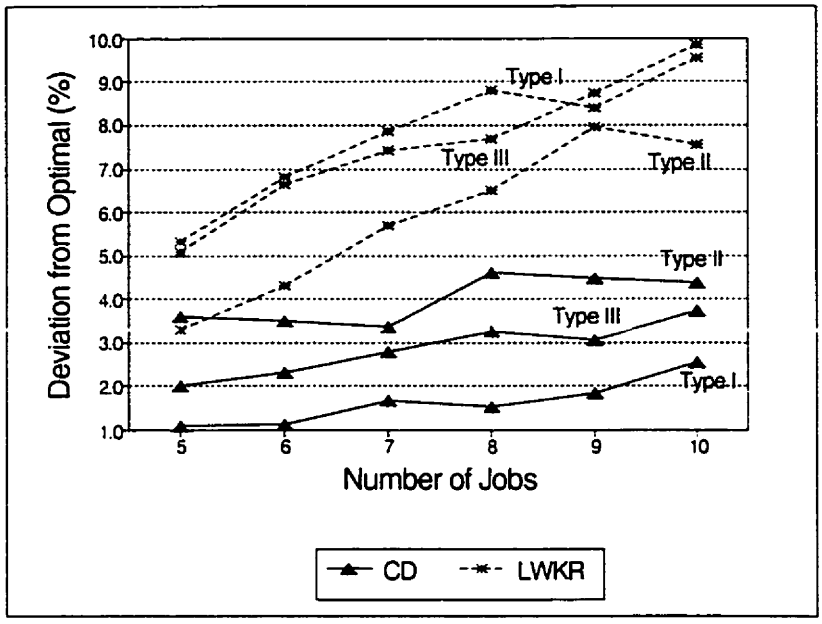


Figure 3.7: Minimizing the mean flowtime with CD and LWKR.

steadily with an increasing number of jobs upto about 25, but levels off with a consistent difference from the LWKR data as the number of jobs increases further. This observation holds for all three configurations, although the CD dispatching still appears at its best for Type I configurations. The results presented in Figures 3.7 and 3.8 generally demonstrate the superiority of CD dispatching.

Another indication of how well a particular method outperforms other methods can be found in the number of times the given method produces the best result in identical test situations. This information is presented in Table 3.10 using, as an example, the case of a pure flowshop (Type I configuration). The CD algorithm is compared in this table with the LWKR rule and a slightly modified NEH algorithm [34]. The NEH algorithm was developed originally to minimize the makespan in

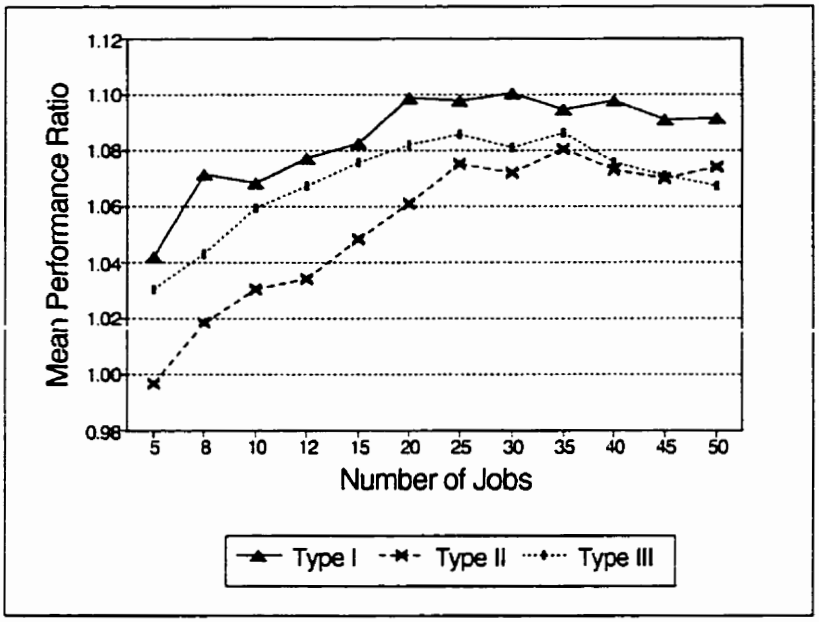


Figure 3.8: Minimizing the mean flowtime in large problems.

pure flowshops. Consequently it can be used only for a Type I configuration. It has been selected here because of the good results reported for it in [35] and [46]. The original algorithm is modified in order that it may serve to minimize the mean flowtime. The change is to simply use initial sequences of jobs that are sorted in SPT order, rather than the EDD order. The remaining comparisons that are presented in Table 3.10 for the Type II and III configurations are between CD and LWKR dispatching only. Of course, the modified NEH-based method does not apply to these two configurations. The CD approach is seen from Table 3.10 to be comfortably better than the LWKR and SPT dispatching rules for all three configurations.

Table 3.10: Number of times best solution found.

	CONFIGURATION							
	Type I			Type II		Type III		
# Jobs	CD	LWKR	NEH	CD	LWKR	CD	LWKR	SPT
5	64	32	100	85	79	73	34	46
8	51	3	78	82	51	77	12	24
10	38	1	79	79	34	82	9	20
12	36	0	77	83	30	95	0	16
15	34	0	77	91	20	93	3	14
20	46	0	66	104	6	102	4	4
25	28	0	82	107	3	105	2	3
30	39	0	71	108	2	100	4	6
35	35	0	75	109	1	103	6	1
40	34	0	76	109	1	107	2	1
45	24	0	86	109	1	104	6	0
50	26	0	84	109	1	102	8	0

3.4.3 Minimizing the Mean Tardiness

Figure 3.9 presents the results when the CD algorithm is used to minimize the mean tardiness. In this instance, the Modified Due Date dispatching rule (MDD) is used for comparison because its effectiveness has been demonstrated in flowshop situations [46]. The comparison that is given in Figure 3.9 generally shows the same trends as those exhibited in Figure 3.7, except at a higher 6 to 8 % deviation from optimal. Nevertheless, the superiority of the CD approach to MDD dispatching is clearly visible.

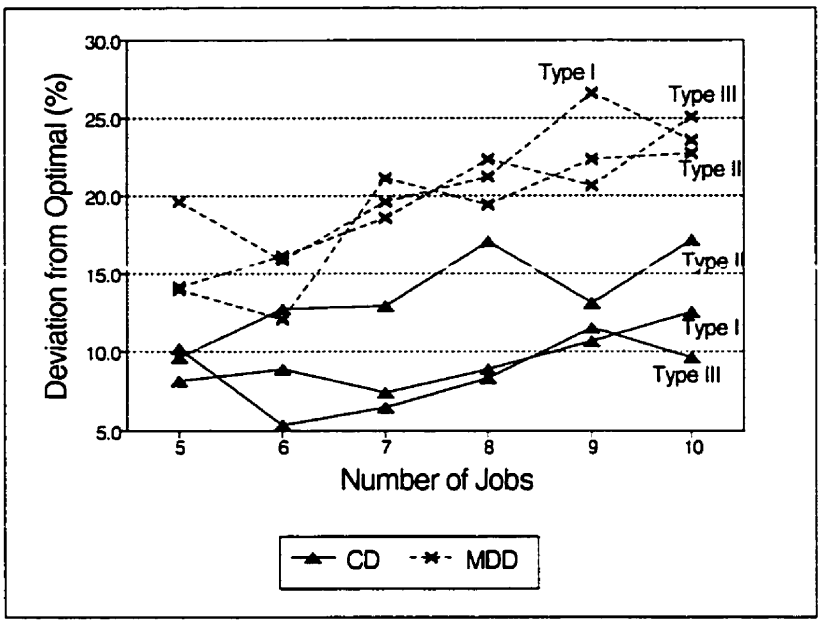


Figure 3.9: Minimizing the mean tardiness with CD and MDD.

When minimizing the mean tardiness, CD uses dynamic programming to optimize the core sequences. This is acceptable when the number of jobs is not more than about 15 to 20. If a greater number is involved, then the computational requirements of finding optimal core sequences become overwhelming. Obviously, near-optimal solutions that can be generated quickly for large problems can be substituted for optimized core sequences. The effect of non-optimization in the core sequence is discussed later in this chapter. The performance of CD in minimizing the mean tardiness in flowshops involving a large (up to 50) number of jobs will be addressed in the next chapter, after a new method for finding near-optimal core sequences is presented.

3.4.4 Minimizing the Number of Tardy Jobs

In order to further test CD's flexibility of application for different criteria, its performance in minimizing the number of tardy jobs in flowshops is evaluated. This time, CD finds the optimal core sequences by employing Hodgson's form (see Baker [8]) of the algorithm developed by Moore [67] to minimize the number of tardy jobs in a single machine. The behavior of the more common dispatching rules in minimizing the number of tardy jobs in flowshops and jobshops is not well documented. Consequently, a series of tests was undertaken to compare the EDD, SPT, MDD and LWKR rules when applied to flowshops having any of three configurations considered in this thesis. The tests revealed a slight advantage in favor of using the SPT rule when the number of jobs is less than about twenty. When the number of jobs is greater than that, the MDD rule becomes attractive. Therefore, it is decided to evaluate CD in comparison to the SPT rule when comparing problems with small job sizes, and to the MDD rule for large sized problems.

Figure 3.10 shows the results obtained from using CD for minimizing the number of tardy jobs, expressed as percentage deviation from the optima. The results that are obtained by using the SPT rule are also included in Figure 3.10 for comparison. The indications from Figure 3.10 are that, even though CD's performance is significantly sub-optimal, it is nonetheless appears somewhat better than that of the SPT rule.

CD dispatching becomes substantially superior as the number of jobs increases. This is evident from Figure 3.11, where CD's performance is compared with the MDD rule in terms of PR. MDD is the reference method. It may be concluded,

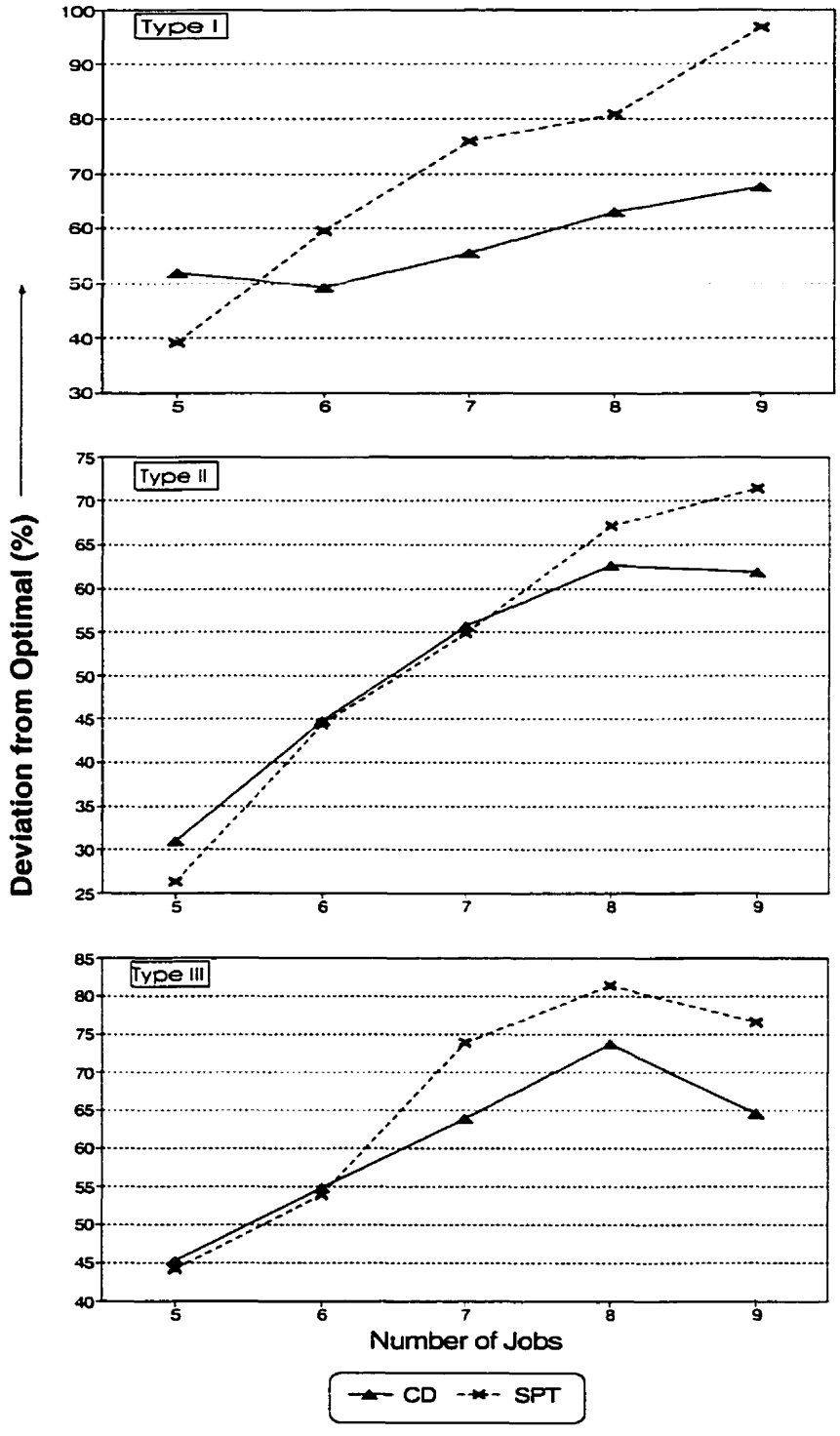


Figure 3.10: Minimizing the number of tardy jobs.

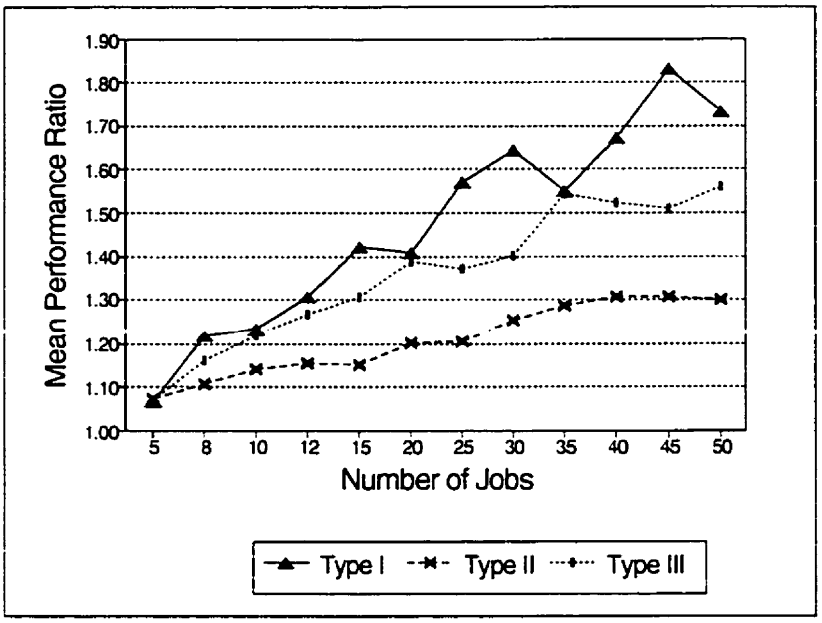


Figure 3.11: Minimizing the number of tardy jobs in large problems.

therefore, that CD is more powerful than traditional dispatching rules for minimizing the number of tardy jobs in flowshops.

3.4.5 Non-FIFO Buffers

Thus far, the evaluation of the CD algorithm has concentrated on the aspect of loading the FMC. The ability to select from the jobs waiting in a queue was assumed to take place only at the buffers holding jobs not yet started. The CD approach can also be applied, like other traditional dispatching rules, to any machine in the cell where a selection of parts from a buffer, including in-process parts, is possible. The types of buffers in the FMC can be classified into two distinct types: FIFO and non-FIFO (or unconstrained) buffers. In FIFO buffers, the parts are prioritized

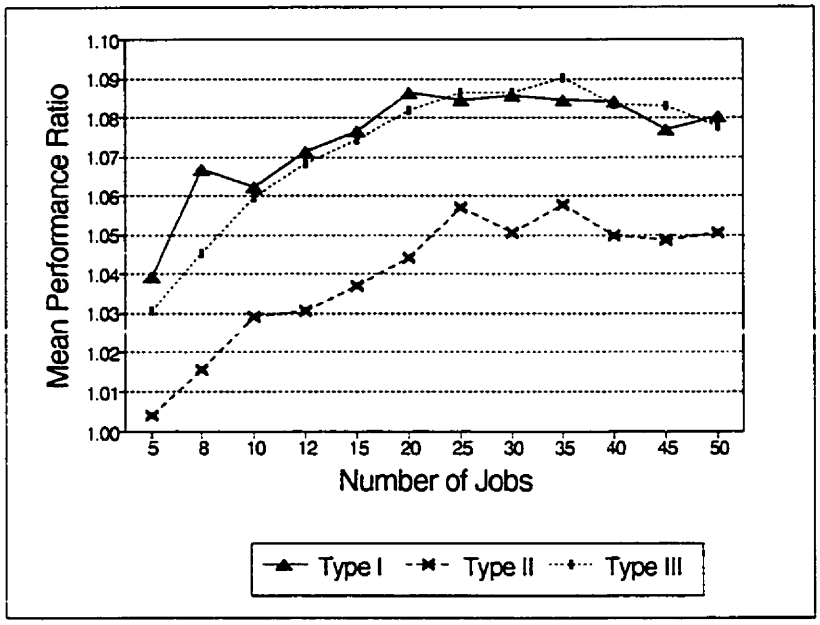


Figure 3.12: CD performance for mean flowtime and non-FIFO buffers.

on a first-come, first-served basis only. In non-FIFO buffers, on the other hand, there is a possibility to select any one of the parts currently residing in the buffer for immediate processing. The CD algorithm is tested next in cases where all the buffers in the cell are unconstrained. The comparisons are performed, as previously, against the LWKR and MDD dispatching rules.

Figure 3.12 displays the results of minimizing the mean flowtime when all the buffers are non-FIFO. They are consistent with those found in Figure 3.8, albeit with lower PR values. The results of Figures 3.8 and 3.12 demonstrate the flexibility of CD by showing that it performs consistently, regardless of the part selection constraints in the intermediate buffers.

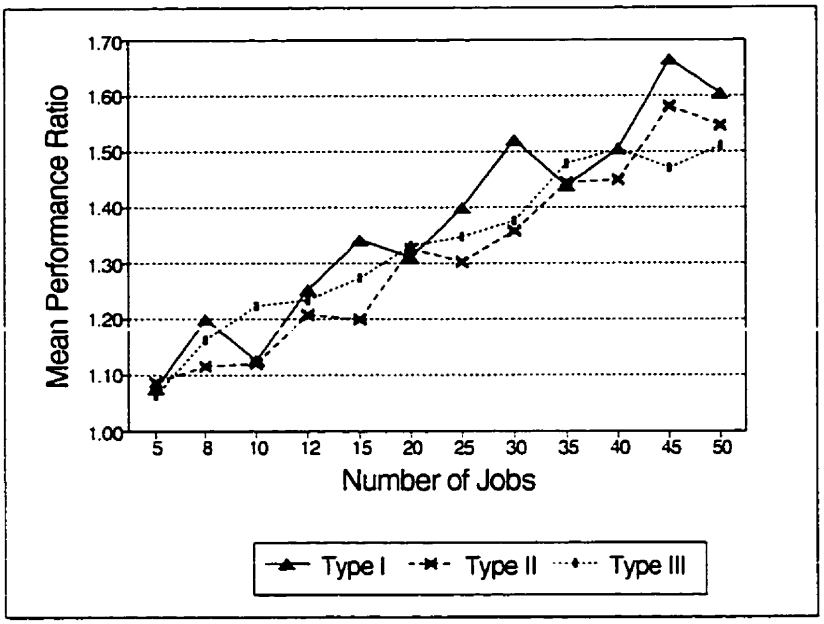


Figure 3.13: CD performance for number of tardy jobs and non-FIFO buffers.

Similar results are obtained when minimizing the number of tardy jobs in flowshops having unconstrained intermediate buffers throughout. Figure 3.13 shows that CD maintains a significant, although slightly decreased, superiority over the MDD rule when all the buffers in the three configurations tested are not restricted to FIFO policies.

3.4.6 Effect of Routing

The results obtained thus far appear to confirm that the traditional dispatching rules become generally less effective as the routings in the flowshop assume characteristics closer to the pure flowshop. This observation is reflected in the results for the Type I configuration, which is a pure flowshop. In the Type I configuration, CD

outperforms the other dispatching rules by a margin that is greater than in the other two configurations, which allow two different job routes. The fact that CD's relative performance is better in the pure flowshop is an indication of its ability to respond to routing limitations in the shop. This feature is an important one because the status of a cell is changing constantly during processing and, at any instant, certain routings may predominate. For example, consider the Type III configuration in the event when an inordinate number of the jobs have merely one of the two possible routes (say route M1 - M3 - M4). This situation resembles the pure flowshop closely, even though the physical configuration is strictly not a pure flowshop. An effective scheduling system should be able to adjust to the new conditions, even though they may be temporary. Based on the results shown in Figures 3.7 through 3.11, CD dispatching indeed appears to possess the adaptability needed for accommodating such dynamic situations as they may arise.

3.5 Optimization in the Core Sequence

The basic premise used in constructing the SC matrix is that each machine attempts to have its candidates selected for dispatching by minimizing its sequence costs (SC_{ky}). This minimization is achieved through optimally sequencing the jobs in the core. To investigate the effect of non-optimal sequences in the cores, an experiment is performed that involves CD for minimizing the mean flowtime. This criterion is selected because the optima can be calculated simply for large sized problems. In this experiment, LPT and randomized sequencing are used in Equation (3.7) rather than the (optimal) SPT. LPT is used because it works to maximize rather than minimize the mean flowtime. This will show the effect when all machines

jointly perform sub-optimally. The resulting performance of the CD algorithm is evaluated by using 1100 test problems generated randomly, again using the method of [51]. Figure 3.14 displays the results, which are expressed as a percentage deviation from the results achieved with optimized core sequencing. The results show that, with randomized sequencing in the core, the CD performs at about 8 to 10% worse than with the optimized cores. Interestingly, when LPT is employed in the cores, the performance of CD is within only 1% of the results from CD with optimized cores. Figure 3.14 show that the impact of a non-optimized core is lessened if the sub-optimal method is applied uniformly to all the core sequences. In other words, with a 'level playing field' in which all the machines are subject to the same rule, the CD algorithm appears to be not very sensitive to the rule used in the core. Nevertheless, Figure 3.14 demonstrates that optimized core sequencing provides a solution that is generally better than the non-optimized one. Although the difference is small, approximately two out of three test problems in the data sets had a better solution when the core sequence was optimized.

A similar experiment to determine the impact of optimization in the core sequences is done by testing CD, with different core sequence generation methods, to minimize the number of tardy jobs. The test data that is used is the same data that gave the results shown in Figure 3.11. The configuration that is used is Type I, and it is selected arbitrarily.

The experiment involves running the CD algorithm for the test data three times, each time using one of the EDD, SPT and MDD priority rules in place of Hodgson's algorithm to determine the core sequences. The resulting performance of CD in each

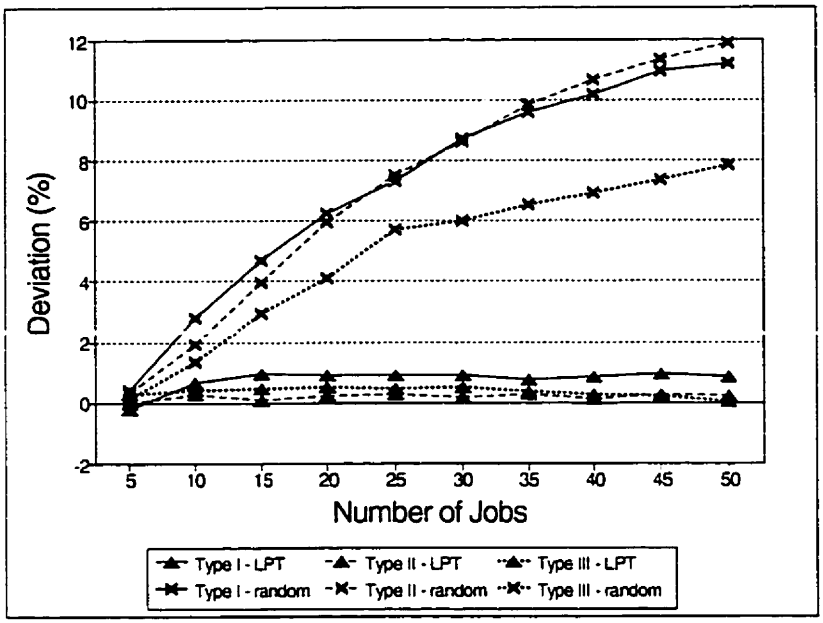


Figure 3.14: Performance of CD with non-optimal core sequencing

instance is shown in Figure 3.15. The results from CD with Hodgson's algorithm are also shown in Figure 3.15 for comparison.

Figure 3.15 clearly demonstrates that using other than the optimal sequence in the core can significantly degrade CD's performance. Of the three priority rules used in the experiment, the MDD rule was the better one, yet it resulted in about a 17% increase in the total number of tardy jobs, for the same data set, as compared to when Hodgson's rule is employed. Hence, it may be concluded that optimal sequencing in the core can be expected to have a positive impact on CD's performance. The significance of that impact may vary, depending on the performance criterion.

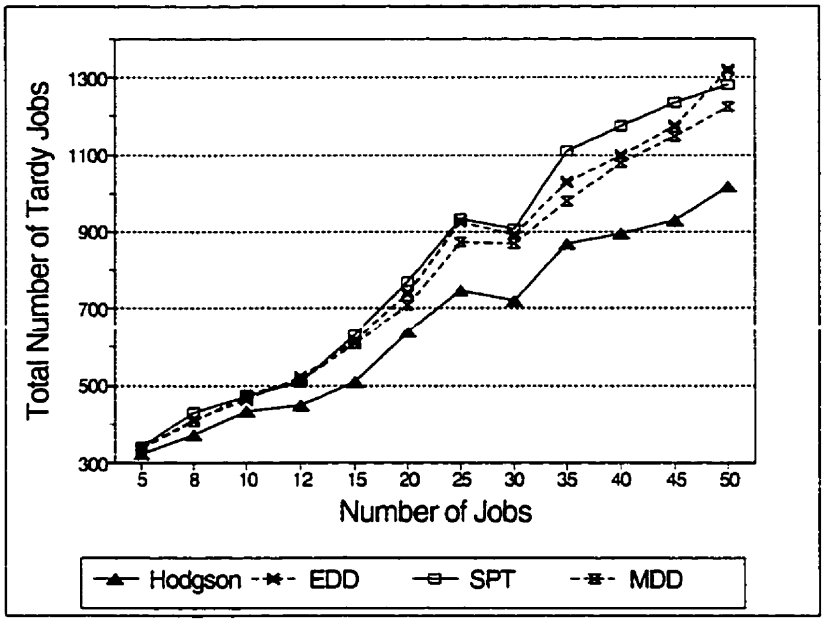


Figure 3.15: Effect of different core sequences on CD's performance.

3.6 Conclusions

Cooperative Dispatching, CD, was introduced in this chapter as a new scheduling procedure that is suitable for FMCs. It is designed to have a high flexibility in its applications, and computer simulations were performed for the purpose of evaluating this flexibility. The simulations involved a comparison of CD with a number of other traditional dispatching rules. The comparisons covered three flowshop configurations, and three different performance criteria: 1) minimizing the mean flowtime; 2) minimizing the mean tardiness; and 3) minimizing the number of tardy jobs. The computer simulations, which involved randomly generated test problems, led to three main conclusions regarding the performance of CD.

First, CD exhibited a performance that was consistently better than the alternative dispatching rules it was compared with in all the three flowshop configurations. The results indicated a low sensitivity of CD to the differences between the three configurations. This result reinforces CD's claims of flexible application in flowshops having different layout configurations (or job routings).

Secondly, CD was able to accommodate quite well flowshops where intermediate buffers were not restricted to a FIFO policy in servicing the waiting jobs. This was in contrast to the other dispatching rules, whose performance stood to benefit in proportion to the number of intermediate buffers that were not restricted to FIFO queueing policies. This fact again demonstrates CD's flexibility under different hardware constraints.

Thirdly, CD showed a good ability to meet a number of different performance criteria, simply by using the appropriate single machine sequencing problem to help the individual machines collectively reach dispatching decisions. In all the computer simulations, and for all of the performance criteria that were considered, CD emerged stronger than the best of the other dispatching rules to which it was compared. Together, these three conclusions are a strong indication of CD's overall scheduling flexibility. This flexibility means minimal disruption, particularly the need for re-writing of software, in the event of reconfiguration or other managerial changes in the system.

The third conclusion implies that CD is likely to perform well for a given performance criterion, provided that a fast method is available for determining optimal

solutions in a single machine setting for that criterion. It was already pointed out that, for some criteria such as minimizing the mean tardiness, optimal solutions may be difficult to obtain because of the extreme computational requirements when the number of jobs is large. In order to address such complications, a new method for sequencing n jobs on a single machine is introduced in the following chapter. This new method employs artificial neural networks. It is designed to provide fast and near optimal solutions for generalized performance criteria. The neural networks work to support the calculations for generating the SC matrix when the performance criteria are NP-hard, like the minimization of the mean tardiness.

Chapter 4

Single Machine Sequencing with Neural Networks

4.1 Introduction

The core equation (Equation 3.7) described in subsection 3.2.2.3 of the preceding chapter is a single machine sequencing problem. As noted in Chapter 3, this problem is NP-hard for some performance criteria, and most of the good search heuristics or optimization algorithms in those cases consume excessive computational time. The heuristics that are fast are generally designed for specific criteria and they are not usually portable to other performance criteria. In this chapter, an alternative approach that is based on artificial neural networks is presented. The choice of neural networks is motivated by two factors: speed and flexibility. Computational speed facilitates the development of algorithms for other types of problems where a large number of sub-problems having the form of a single machine sequencing problem need to be solved quickly. The CD approach described in the preceding chapter is one such example. The second factor is flexibility with respect to the performance criteria. In reality, the performance criterion may not be a standard one but, rather, a complex cost function for which no quick and accurate heuristics are known. In such situations, either a customized algorithm has to be developed or 'quick and dirty' sorting procedures may be employed (Potts and Van Wassenhove [51]). Sorting in the order of non-decreasing due dates (the EDD rule) or non-decreasing processing times (SPT rule) are examples of the latter. Hence, there

is a need for an approach that quickly produces good sequences when the performance criteria are uncommon, but without having to develop a unique algorithm or heuristic for every individual situation.

The method for solving single machine job sequencing problems that is presented here is based on artificial neural networks (ANN). It is conceptually simple. An ANN is used to learn a functional relationship between a set of example, single machine problems and the corresponding sequences that optimize the given performance criterion. This 'trained' neural network is then able to apply the learnt relationship to new problems through its generalization property.

4.2 Artificial Neural Networks

An artificial neural network is a collection of highly interconnected processing units that has the ability to learn and store patterns, as well as to generalize when presented with new patterns. The 'learnt' information is stored in the form of numerical values, called weights, that are assigned to the connections between the processing units of the network. The type of neural network that is used here is called a 'feedforward' network. It organizes the processing units into multiple layers, one of which serves as an input layer, and another as the output layer. The rest of the layers are called 'hidden' layers. They exist between the input and output layers. Figure 4.1 illustrates a three-layer feedforward neural network. The units in the input layer serve as a terminal for receiving the input patterns. These units are clamped to the values contained in the patterns introduced at the input layer.

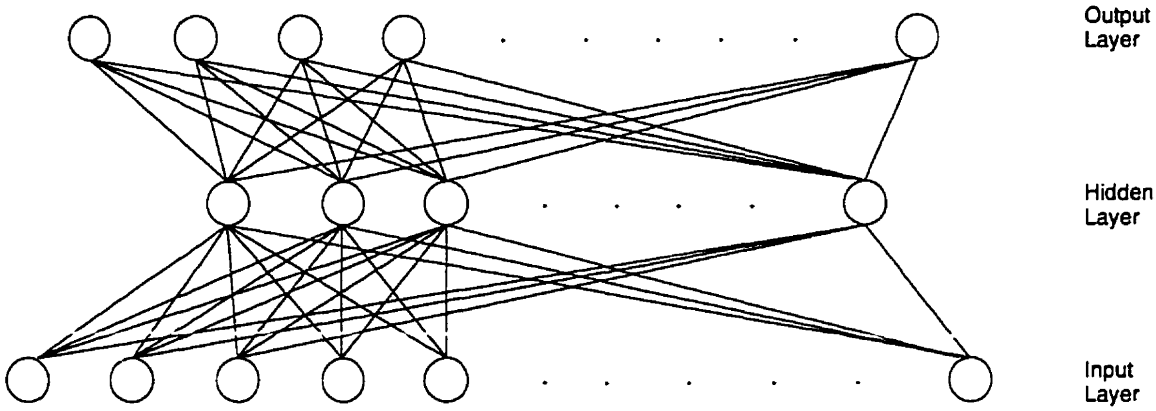


Figure 4.1: A three-layered, feedforward neural network (BPN).

The units in the output layer hold the values resulting from the feedforward computations that are triggered after the network is stimulated by the introduction of a pattern at the input layer. A neural network is said to 'recall' a pattern when it responds with an output to an input stimulus.

The network shown in Figure 4.1 is trained by using a supervised learning mode. During supervised learning, the weights of the inter-layer connections between the units are modified incrementally until the network is deemed to have 'learnt' the relationship between the set of input patterns and the corresponding outputs. The most commonly used method for modifying these weights is the backpropagation algorithm of Rumelhart and McClelland [68]. In this algorithm, a pattern is applied at the input layer, and the stimulus is propagated forward until final outputs are calculated at the output layer. These outputs are compared with the desired result for the pattern considered, and the errors are computed. The errors are then propagated backwards through the network as a feedback to the preceding layers

to determine the changes in the connection weights that will minimize these errors. A series of such input-output training examples is presented repeatedly to the network during the backpropagation algorithm until the total sum of the squares of the errors is reduced to an acceptable level. At this point, the network is considered 'trained'. Data presented at the input layer of a trained network will result in values from the output layer that are consistent with the relationship learnt by the network from the training examples. A feedforward neural network that is trained by using the backpropagation algorithm, is often termed a Backpropagation Network (BPN). The backpropagation algorithm is implemented with the use of a simulation program written in the DESIRE/NEUNET matrix language [69]. The design, training and testing of all the neural networks described in this chapter is done with the use of this software package.

4.3 A Neural Network for Single Machine Sequencing

The neural network that is proposed for the single machine sequencing problem has the same architecture as that shown in Figure 4.1. It consists of an input layer of eleven units, one hidden layer, and an output layer having a single unit. The number of units in the input and output layers is dictated by the specific representation adopted for the sequencing problem. In the proposed representation, each of the n jobs in the problem are described by an 11-tuple vector of continuous values. Each vector, or pattern, holds information particular to the job it represents as well as to its relation to the other jobs in the problem. Therefore, the input layer has eleven input units to accommodate each one of the elements of the input vectors. This particular choice for representing the problem, whereby the jobs are processed by

the neural network one job at a time, is dictated to a large degree by the necessity of avoiding a dependency between the trained network and the number of jobs in the sequencing problem. Individual presentation of the jobs at the input layer allows the trained neural network to process sequencing problems without restriction to the same problem size as that used in the training examples. Each element of the input vector holds a specific piece of information, detailed as follows:

$$\text{unit 1} = p_i/M_p \tag{4.1(a)}$$

$$\text{unit 2} = d_i/M_d \tag{4.1(b)}$$

$$\text{unit 3} = SL_i/M_{sl} \tag{4.1(c)}$$

$$\text{unit 4} = h_i/10.0 \tag{4.1(d)}$$

$$\text{unit 5} = t_i/10.0 \tag{4.1(e)}$$

$$\text{unit 6} = \bar{p}/M_p \tag{4.1(f)}$$

$$\text{unit 7} = \bar{d}/M_d \tag{4.1(g)}$$

$$\text{unit 8} = \bar{SL}/M_{sl} \tag{4.1(h)}$$

$$\text{unit 9} = \sqrt{\frac{\sum (p_i - \bar{p})^2}{n \times \bar{p}^2}} \tag{4.1(i)}$$

$$\text{unit 10} = \sqrt{\frac{\sum (d_i - \bar{d})^2}{n \times \bar{d}^2}} \tag{4.1(j)}$$

$$\text{unit 11} = \sqrt{\frac{\sum (SL_i - \bar{SL})^2}{n \times \bar{SL}^2}} \tag{4.1(k)}$$

where

p_i = processing time for job i.

d_i = due date for job i.

$SL_i = (d_i - p_i)$.

M_p = longest processing time among the n jobs = $\max[t_i], \quad i \in n$.

M_d = latest due date among the n jobs = $\max[d_i], \quad i \in n$.

M_{sl} = largest slack among the n jobs = $\max[SL_i], \quad i \in n$.

The input vectors have literal interpretations. Consider, for example, the input vector given in Table 4.1. This vector represents one of the jobs in a given problem. Unit 1 shows that the vector in Table 4.1 describes a job that has the longest processing time of the jobs to be sequenced. Units 2 and 7, together, indicate a much tighter than average due date. That no slack is available for this job can be gathered from the value of unit 3. On the other hand, unit 9 indicates that the job will be competing in an environment characterized by a significant variability in the processing times of the jobs. The holding and tardiness costs are in the ratio of 3:4 for the job represented by the above vector, tardiness being the costlier. During training, the neural network is taught where a job having these characteristics is best located in the final sequence for the performance objective that is used.

Table 4.1: Example of an input vector for a job.

unit	1	2	3	4	5	6	7	8	9	10	11
value	1.00	0.27	0.00	0.30	0.40	0.45	0.60	0.52	0.71	0.34	0.55

The output unit in the proposed neural network assumes values that are in the range of 0.1 to 0.9, the magnitude being an indication of where the job represented at the input layer should lie in the sequence. Low values suggest lead positions in the sequence; higher values indicate less priority and, hence, positions towards the end of the sequence. The number of units in the hidden layer is selected by trial and error during the training phase. Hidden units have the role of identifying the presence or absence of features in the input data that are relevant to the solution

of a problem. Those hidden units that consistently assume near zero values, when recalling patterns, are probably redundant. Therefore, they may be pruned with little or no loss in the network's overall performance. The numbers of units in the input, hidden and output layers respectively are often used to describe a network's structure. For example, a network having eleven input units, 10 hidden units and one output unit is referred to as a 11-10-1 network.

4.3.1 Training Methodology

The neural network is trained by presenting it with a pre-selected set of input-target patterns. The input training patterns are the 11-tuple vectors, extracted from a population of n-job, example problems. The target associated with each training pattern is a value that indicates the position occupied in the optimal sequence by the job represented by the input pattern. The target value, $G_i(S)$, for the job holding the i^{th} position in the optimal sequence is determined as:

$$G_i(S) = 0.1 + 0.8\left(\frac{i-1}{n-1}\right), \qquad i = 1, \cdots, n. \tag{4.2}$$

Equation (4.2) ensures that the n target values are distributed uniformly between 0.1 and 0.9. It is needed to accommodate the form of the sigmoidal activation function that is employed in the backpropagation learning algorithm [68] and whose outputs are, in the limit, between 0 and 1. The range 0.1 to 0.9 is used instead so that the target values assigned near the function's limits are theoretically attainable.

The number of jobs, n, in the example (training) problems has a bearing on the target value that is ascribed to each input pattern. As the output space ranges from 0.1 to 0.9, the n target values needed by the n jobs in the problem are spaced equally in increments of $(0.9 - 0.1)/n$ within that range. Consequently, as n grows, the target values for jobs in the sequence tend to migrate closer together. This means that the targets are more accurate in specifying the desired position in the sequence. The tighter tolerances may make training more difficult but, on the other hand, the trained network is able to assign the job positions with better precision in a sequence. Therefore, training patterns should preferably come from example problems having n as large as possible. This is not always easy to accomplish because finding optimal solutions for NP-hard example problems is increasingly more demanding, in terms of computational storage and time requirements, as n increases. Hence, a trade-off needs to be made between the size of n for the example problems and the time and resources that can be afforded for the training phase. Where optimal solutions are obtained by means of complete enumeration, 8-job problems are chosen here as the source for the training patterns. This is because complete enumeration for problems involving more than 8 jobs requires significantly more computer time due to the number of individual problems that need to be solved to generate the training set. In instances where optimal solutions can be found by dynamic programming (Held and Karp [65]), a procedure which is more efficient than complete enumeration, the training patterns are extracted from 12-job rather than 8-job example problems. Although dynamic programming could have been used to extract the patterns from problems having n as high as 20, the decision to use 12-job problems is influenced by expediency, computer resources and development time limitations, as well as the ease of making comparisons.

The steps for training and employing the neural network for the single machine sequencing problem are given below.

(a) Generate a random set of example problems.

(b) Find the optimal solutions for the example problems.

(c) Select the input-output training patterns from the solved problems.

(d) Train the neural network by using backpropagation.

(e) Use the trained neural network to solve new problems.

The n-job example problems are generated randomly by using Potts and Van Wassenhove's method [51]. The processing time for the i^{th} job, p_i, is generated from the uniform distribution [1,100]. The total of the processing times of the n jobs, $P = \sum_{i=1}^{n} p_i$, is computed. The due date for each job (d_i) is then selected randomly from the uniform distribution [P(1-TF-RDD/2),P(1-TF+RDD/2)] where RDD is a parameter representing the range of due dates and TF is a tardiness factor. RDD and TF assume combinations of values between 0.1 and 1.0 for the problem domain considered. Only a quarter of the n jobs in any example problem are selected, at random, to serve as training patterns. This restriction is artificial but it enables the neural network to be exposed to the characteristics of more problems whilst remaining within the software's storage capacity. Hence, the total number of example problems required to generate Y training patterns is $4Y/n$. The overruling guideline for determining Y is to employ the maximum number of training patterns possible. This number depends, again, on factors related to the software and the available computing resources. What is important, however, is that a sufficient number of training patterns are used. One way of determining the minimum number of inde-

pendent training patterns that are needed is to start with a limited number, and then increase this number gradually in separate training trials. When it is observed that the addition of further patterns does not significantly reduce the generalization error, then it may be accepted that the lower bound for the required number of patterns has been surpassed. This approach is implemented in the training of the neural networks described in this Chapter. In all instances, the number of patterns used in training a neural network is well above the lower bound, as determined by trial and error, for the case under consideration.

Test problems that are generated randomly in a manner similar to the training set, but with a different seed, are used during the training phase to evaluate the 'learning' of the neural network. This evaluation is based on monitoring the average 'positioning error'. The positioning error, e_q , for pattern q is measured as :

$$e_q = \frac{| O_q - G_q | \times n}{0.8} \quad . \tag{4.3}$$

where

O_q = output response when pattern q is presented at the input layer; and

G_q = target response for test pattern q.

Equation (4.3) indicates how closely the neural network is able to position the job represented by pattern q to the position that the job should occupy in the optimal sequence. An e_q that is larger than unity shows that the error is greater than one full position in the sequence. Conversely, an average error for the test data patterns that is less than unity suggests that the neural network is able, on average, to place the jobs in their correct sequential positions.

During the training stage, training is interrupted intermittently and the network is tested by using the set of test patterns. The average positioning error is compiled and examined at each interruption. This process is ended when the error just begins to increase with further training. This is because the network then starts to 'memorize' the training set which leads to a loss in its generalization abilities.

4.3.2 Evaluation of Learning Capability

The effectiveness of the proposed neural network methodology for job sequencing is evaluated next by testing with a performance criterion for which a simple solution is known to exist. This criterion is the minimization of the maximum job lateness. It is optimized by merely sorting the jobs in the order of non-decreasing due-dates [8]. Hence, the target values, $G_i(S)$, are computed for a sequence S sorted in EDD fashion. The objective in this evaluation is to determine how effective the neural network is in deducing, based solely on its exposure to the training problems and their solutions, that the EDD sorting rule optimizes the criterion.

The 4320 training patterns are derived from 1600, 12-job example problems generated randomly for combinations of TF and RDD that range, in increments of 0.1, between 0.2 and 1.0 for TF, and between 0.1 and 1.0 for RDD. 12-job problems are used to find how a neural network, which is trained more conveniently from example problems having a relatively small n (i.e. $n = 12$), performs when dealing with realistic problems having a much higher value of n. This is not an issue for the performance criterion currently under consideration because optimal solutions to large

sized problems can be obtained very conveniently. However, there are other criteria for which finding the optimal solutions for the training data does pose difficulty as the problem size increases. For those cases, the use of training problems having a 12-job size may provide a satisfactory balance between size and computational time requirements.

Initially, a 11-5-1 network is used. To determine how closely the sequences generated by the neural network for the test set match the correct sequences, a 'distance' measure, TD, is used in addition to the average positioning error, e_q. TD simply sums the job displacements. A displacement for a job is the number of positions by which it is out of sequence. For example, if a job is out of its optimal location by two positions, then its displacement is two units. The lower is the total displacement, the closer are the jobs to their optimal positions in the sequence.

Figure 4.2 shows the behaviour exhibited during the training of the 11-5-1 network. As the training progresses, TSS (the total sum of the squares of the errors) decreases as the network learns the set of the training patterns. After every 1000 cycles, training is interrupted and the test patterns are processed through the network to calculate e_q and TD. The test data contains 300 patterns representing 25, 12-job problems that are generated in a random manner identical to that used for the training set, but with a different seed.

Figure 4.2 shows that the training of the 11-5-1 network can be terminated after 11,000 cycles, when e_q is seen to be minimal. At that point TD is observed to be zero, meaning that the network has sequenced all of the jobs optimally.

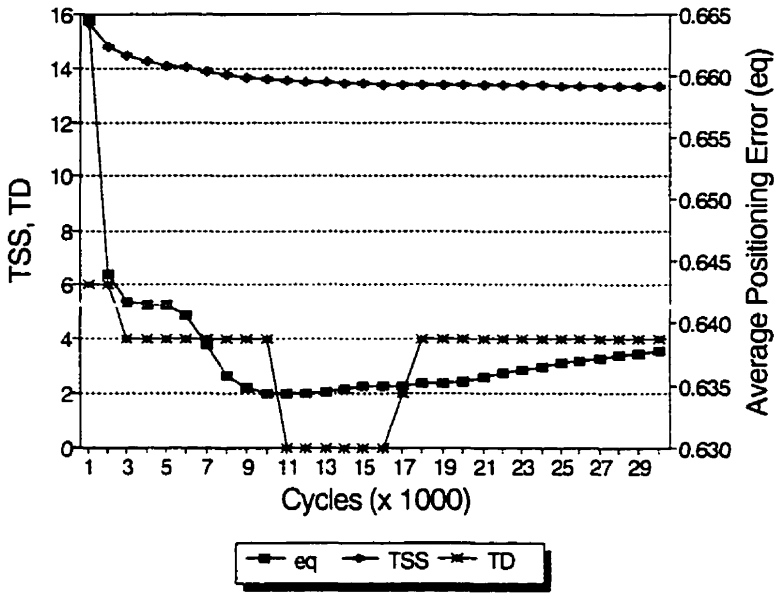


Figure 4.2: Training of 11-5-1 BPN.

The entire training process just described for the 11-5-1 network is repeated separately for networks having seven, nine and twelve units in the hidden layer. Figure 4.3 compares the learning abilities of each of these networks. In the configurations with fewer hidden units, the TSS is observed to converge to higher values. This indicates that the networks with more units in the hidden layer are better able to learn the training data. Generally, the final number of units required in the hidden layer is that number beyond which the addition of a further unit results in only negligible improvement in the network's ability to learn the training data. This trial and error method is the technique that is chosen in this thesis for selecting the number of hidden layer units in the neural networks.

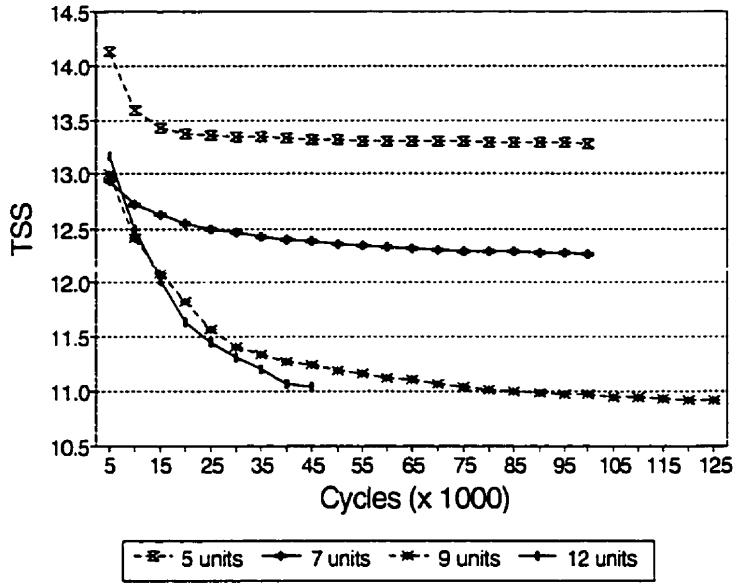


Figure 4.3: TSS during training with different hidden layer sizes.

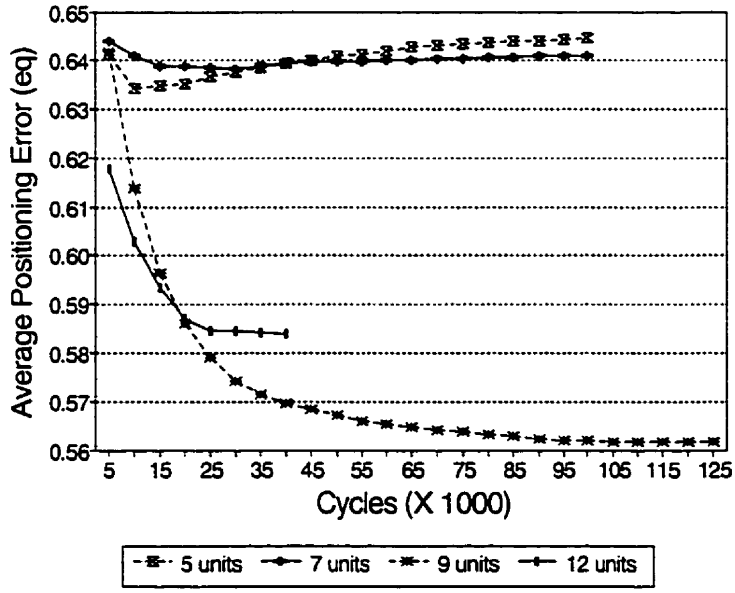


Figure 4.4: Behaviour of e_q with training.

The manner in which this behaviour translates to a generalization capability is illustrated in Figures 4.4 and 4.5. Figure 4.4 compares the average e_q in the four network configurations. It is seen that e_q is less in the 11-9-1 network in comparison with the 11-5-1 and 11-7-1 networks. Moreover, there is no clear advantage in the 11-12-1 network, where e_q is at a level comparable to that in the 11-9-1 network. Figure 4.5 compares the four networks with respect to TD. The 11-5-1 and the 11-7-1 networks are seen to lower the TD in the test data better than the other two networks that have more units in the hidden layers. This means that, although the 11-5-1 and 11-7-1 networks have higher positioning errors as seen in Figure 4.4, they are able to generalize their learning and find correct sequences in new problems better than the other networks that are trained to a lower positioning error. This anomaly is explained by the fact that as a network learns 'better' by means of minimizing the positioning errors, it begins to also learn 'additional' data in its efforts to find more accurate relationships between the input and output data. In the case of minimizing the maximum lateness, this additional data is most probably the job processing times, which is irrelevant in finding optimal solutions. Consequently, an emphasis on minimizing positioning error can lead to overtraining and memorization of the training data at the expense of the desired generalization capabilities. The periodic testing to determine TD during training helps to determine when the network can be accepted as satisfactorily trained.

Although Figure 4.5 shows the 11-5-1 and 11-7-1 networks to be almost equivalent in minimizing TD, the 11-5-1 network is selected after 15,000 training cycles. This selection is based on the fewer number of hidden units, as well as on the fact that the 11-5-1 network actually demonstrates a zero TD. Fewer number of units

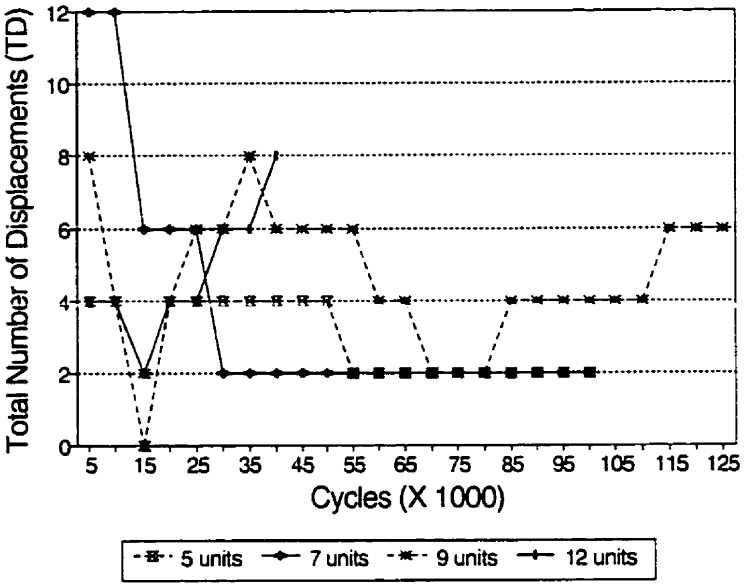


Figure 4.5: Total Displacement (TD) during network training.

in the hidden layer is generally preferable because it results in quicker processing (feedforward propagation) across the network. This 11-5-1 neural network, trained for minimizing the maximum job lateness, is designated MLATENET. All trained networks that assign jobs a position in a sequence, such as MLATENET, will be called neural sequencers. The details of the training of MLATENET and its final weights are given in section A.1 of Appendix A.

4.3.3 Illustrative Problem

An example is presented now to demonstrate how a neural sequencer generates sequences to satisfy the performance criterion. Table 4.2 shows a 7-job problem that serves as an example. This problem is generated randomly for an arbitrarily

selected TF =0.6 and RDD = 0.4. The objective is to sequence the jobs in Table 4.2 to minimize the maximum job lateness.

Table 4.2: Seven-job sequencing problem.

Job	p_i	d_i	h_i	t_i
1	38	206	0	1
2	32	196	0	1
3	96	137	0	1
4	27	116	0	1
5	24	188	0	1
6	87	179	0	1
7	93	228	0	1

The neural sequencer MLATENET will be used here because it is trained already for the performance criterion used in this example. The seven jobs in the problem are converted first into their vector representations by using the set of equations 4.1(a) to 4.1(k). The result of this pre-processing stage is presented in Table 4.3 where the vectors V1 through V7 represent job numbers 1 through 7, respectively. To solve the sequencing problem, each vector is presented individually at the input layer of MLATENET. A feed forward procedure of calculations [68] generates a value between 0.1 and 0.9 that appears at the output unit for each of the seven input vectors. This procedure is like the one employed by Sim et al. [32] in that jobs are processed individually by the network. The objective in [32] is to find the job having the least activation value at the output layer. In the new approach, however, the output value for each job is significant because this output ultimately

determines the job's sequential position in relation to the other jobs of the problem set. A complete sequence is constructed only after each one of the n jobs makes a pass through the network. The output computed by the neural network for each of the input vectors is given in the rightmost column of Table 4.3.

Table 4.3: Problem representation for the example in Table 4.2.

	INPUT UNITS											
JOB	1	2	3	4	5	6	7	8	9	10	11	OUTPUT
V_1	0.40	0.90	1.00	0.00	1.00	0.59	0.78	0.73	0.55	0.20	0.37	0.6937
V_2	0.33	0.86	0.98	0.00	1.00	0.59	0.78	0.73	0.55	0.20	0.37	0.6083
V_3	1.00	0.60	0.24	0.00	1.00	0.59	0.78	0.73	0.55	0.20	0.37	0.2120
V_4	0.28	0.51	0.53	0.00	1.00	0.59	0.78	0.73	0.55	0.20	0.37	0.0980
V_5	0.25	0.82	0.98	0.00	1.00	0.59	0.78	0.73	0.55	0.20	0.37	0.5506
V_6	0.91	0.79	0.55	0.00	1.00	0.59	0.78	0.73	0.55	0.20	0.37	0.4849
V_7	0.97	1.00	0.80	0.00	1.00	0.59	0.78	0.73	0.55	0.20	0.37	0.9043

Sequencing the jobs in the order of the increasing output values results in the job sequence { 4 - 3 - 6 - 5 - 2 - 1 - 7 }, which is an optimal solution.

4.4 Performance for Different Criteria

MLATENET was taught to minimize the maximum job lateness by training with information and solutions based upon 12-job problems. It is important to determine whether MLATENET can generalize the learnt relationship to problems involving different, and especially larger, sizes. In addition, the neural network approach is

evaluated for other commonly used performance criteria to better understand its scope of application.

4.4.1 Minimizing the Maximum Job Lateness

MLATENET is tested by using problems involving upto 100 jobs. Each test consists of 200 replications for each value of n. The test problems are generated randomly by using Potts and Van Wassenhove's method with RF={0.2,0.4,0.6,0.8,1.0} and TF={0.2,0.4,0.6,0.8,1.0}. All h_i are zero and all t_i are equal to 1.0 in the test problems. The results are shown in column 1 of Table 4.4. They are expressed in terms of the percentage deviation from the optimal solution. It is seen from the tests that MLATENET has learnt quite well to sequence the jobs in an order that enables minimization of the maximum lateness.

4.4.2 Minimizing Flowtime Criteria

The neural network approach is evaluated next for minimizing either the mean flowtime or a variant, the weighted mean flowtime. These two objectives are satisfied optimally by sorting the jobs in the order of non-decreasing processing time divided by the weight for each of the jobs (i.e. the WSPT rule)[8]. Optimal solutions, therefore, are easily found for any job size, n. Moreover, one neural network, which is trained to minimize the weighted flowtime, suffices because the mean flowtime is a special case of the weighted flowtime in which all the weights are equal.

Table 4.4: Percentage deviation from optimal for three criteria.

n	Maximum Job Lateness	Mean Flowtime	Mean Weighted Flowtime	CPU Time (ms)
5	0.000	0.009	0.136	0.77
10	0.000	0.016	0.136	1.58
12	0.004	0.018	0.100	1.95
15	0.000	0.016	0.098	2.52
20	0.001	0.016	0.102	3.47
25	0.004	0.017	0.097	4.41
30	0.010	0.016	0.097	5.57
50	0.022	0.016	0.098	10.77
75	0.032	0.016	0.100	18.21
100	0.056	0.015	0.100	28.35

The neural network for minimizing the mean weighted flowtime, MFLONET, is trained in a manner similar to that described for MLATENET. During the generation of the training patterns, however, all the t_i are set to zero. The values for h_i are selected randomly from the uniform distribution [1,10]. Details of the training and testing of FLONET are given in section A.2 of Appendix A.

The trained neural network is assessed initially with respect to the minimization of the mean flowtime. Test data for different job sizes are generated in the same random fashion as the training data except that all h_i invariably equal one. Next, minimization of the mean weighted flowtime is evaluated by using test data gener-

ated identically but with the h_i randomly taking values between 1 and 10. The test data consists of 10 sets. Each set contains 200 problems having size n. The values for n that are tested are identical to those selected for the testing of MLATENET in the previous section.

The results from FLONET are also presented in Table 4.4. The third and fourth columns demonstrate that FLONET has learnt, quite well, the relationship that leads to a minimization of either the mean flowtime or the mean weighted flowtime. This relationship is embodied in sorting the jobs in the order of non-decreasing weighted processing times.

Table 4.4 suggests that, if there is a well structured rule that leads to the optimal job sequence, such as WSPT or EDD sorting, then the neural networks are able to deduce it. Furthermore, even though the training is performed on information from only 12-job problems, the neural networks are still able to apply the learned relationship to problems having n much higher than twelve. Table 4.4 also shows that the CPU time is only milliseconds per problem, when using an IBM compatible, Pentium 150 MHz microcomputer having 16 MB of RAM.

4.4.3 Minimizing the Mean Tardiness

A performance criterion that is studied widely is the minimization of the mean job tardiness (or total tardiness), a problem which is NP-hard [55]. Job due dates are the main influence on the mean tardiness of a given job sequence. The capability of neural networks in sequencing jobs to minimize the mean tardiness is investigated

by training a network using the same training data as that employed for minimizing the maximum job lateness. In this instance, however, the target values for each job are based on sequences that minimize the mean tardiness. These sequences are found by dynamic programming. The details of the training of this network, designated MTARNET, is found in section A.3 of Appendix A. Table 4.5 presents the results from the trained neural network for problems in which n varies between 5 and 12. Each of the test sets contains 200 problems of a size equal to the selected value of n. Also given, for comparison, are the corresponding results from the sorting rules EDD and SPT.

Table 4.5: Percentage deviation from optimal when minimizing the mean tardiness.

n	5	6	7	8	9	10	11	12
MTARNET	6.62	10.93	8.54	8.67	8.14	7.98	6.21	6.37
EDD	24.96	29.46	29.07	32.88	36.71	36.07	38.03	38.58
SPT	6.63	10.14	11.06	11.32	12.54	13.92	13.30	12.92

The neural network's solutions are noticeably more accurate than those obtained from applying the SPT and EDD rules. This observation suggests that the neural network has learned some form of rule, most likely a rudimentary combination of different rules. Nevertheless, there still remains a difference of about 6% to 11% from the optima. This rather significant discrepancy may be traced to the fact that there is no well-defined relationship between a problem's input and the output sequence which the neural network can learn and extend to all the problems in the domain. In fact, there may well be conflicting relationships that impede the learning, causing the network to capture what amounts to an 'average' relationship. The

solution to this bottleneck is to break the problem domain into 'categories', each category containing problems having similar defining characteristics. Such a scheme is described in the next section. Neural solutions can also be improved based upon the implications of Equation (4.3), which relays the positioning error in the trained network. The average positioning error in the network trained to minimize the mean tardiness, for example, is found to be around 1.4. This particular value indicates that any job's current position in the sequence is, on average, nearly one and a half positions away from the preferred position in the optimal sequence. Therefore, the job sequences produced by the neural network appear well groomed, as a result of minimizing the positioning error, for further improvement by means of a simple, adjacent pairwise interchange strategy.

4.5 Neural Job Classification and Sequencing Network

The breakdown of the problem domain into individual categories aims primarily to group elements having similarities. This categorization (or classification) can be performed ideally by competitive neural networks. Competitive networks consist of two layers: an input layer and an output layer. A pattern presented at the input layer stimulates the activation of one and only one output unit. By having as many output units as the number of categories desired, such networks can be trained to classify input patterns into groupings based on similar characteristics.

Baker [8] indicates that the tightness and range for due dates are attributes that determine how 'difficult' or 'easy' it is to generate job sequences to minimize the

mean tardiness. The 'easy' cases represent instances where near optimal results can be achieved by simply sequencing according to either the EDD or the SPT rule. For tardiness-based criteria, a reasonable categorization scheme would be based, therefore, around the average 'tightness' of the job due dates. Such a scheme, centered around a hybrid categorization-sequencing strategy, is developed next. It is illustrated by using the minimization of the mean tardiness as an example. The problem domain is identified first. In this instance, the domain is taken to include all the problems that can be generated for values of TF from 0.4 to 1.0 and RDD values between 0.1 and 1.0. Problems arising from TF that are less than 0.4 are excluded from the domain only because the 'looseness' of the due dates results in comparatively small tardiness values. The selected domain is classified into a pre-specified number of categories. Any new problem would need to be classified initially into one of these existing categories. This procedure is performed by a competitive neural network, called here a Classifier Network, which is specialized in problem categorization.

Figure 4.6 shows how the sequencing hierarchy operates. Data from a given problem is posed to the Classifier Network, which proceeds to categorize the problem into one of the existing categories. After classification, the problem data is transmitted to a neural sequencer that is trained specifically for problems of that particular category. The system depicted in Figure 4.6 is called a Neural Job Classification and Sequencing System, NJCASS.

The Classifier Network is trained by using an unsupervised learning mode. This is accomplished by means of a competitive learning algorithm [70] that modifies

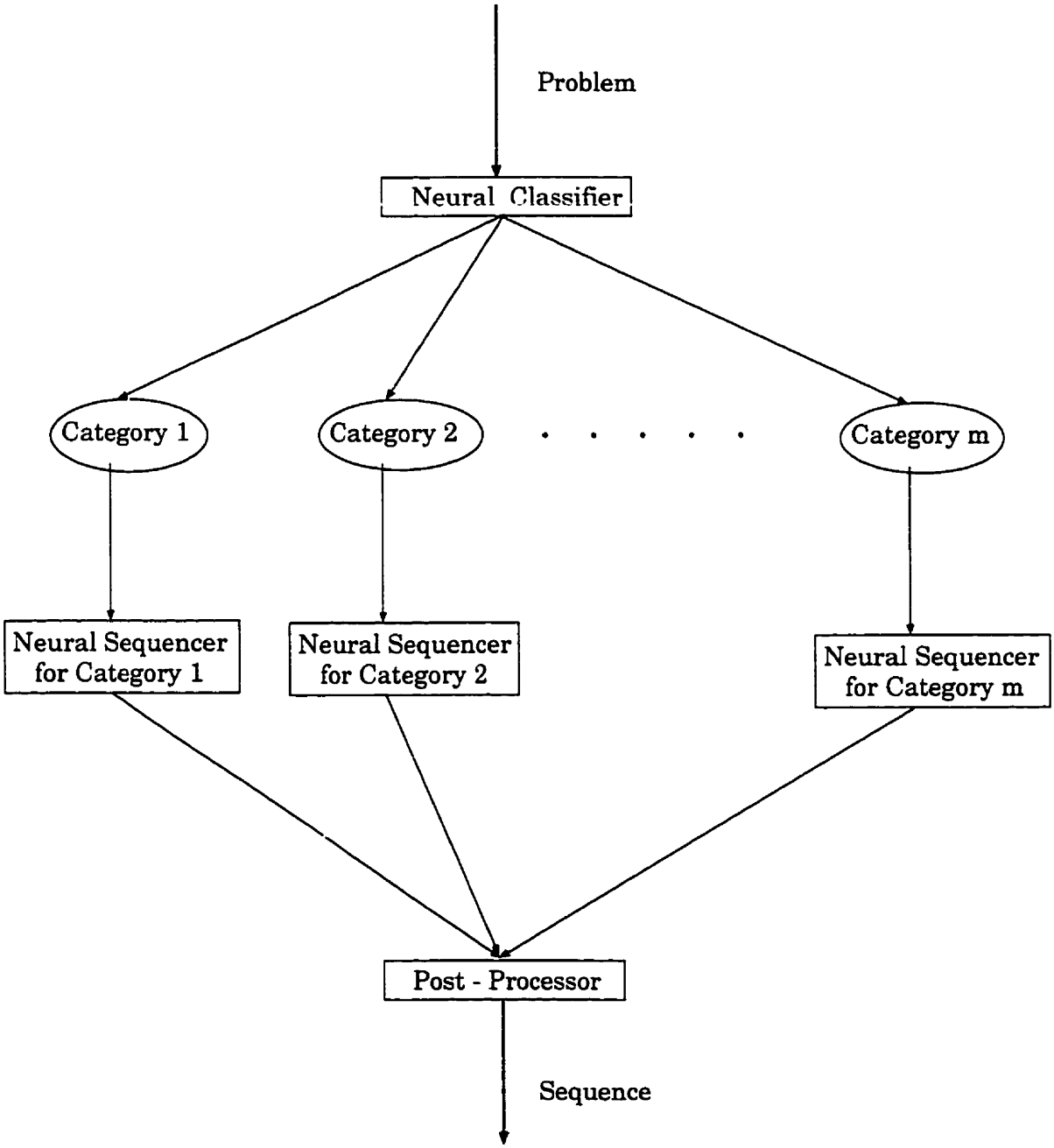


Figure 4.6: Schematic of NJCASS procedure.

the weights so that only one of the output units responds to the input. Whenever a pattern is presented at the input layer, the output unit whose weight vector is closest to the pattern 'wins' the competition. The weight connections leading to the winning unit are updated to reinforce the relation between the input pattern and that unit at the expense of the other units. Inhibitory connections exist between the units in the output layer to ensure that only the 'winning' unit activates. The result is that the input data is organized automatically into a number of clusters, each cluster corresponding to similar inputs and represented by one of the output units. The Classifier Network for minimizing the mean tardiness is called MTCLASS. It is trained by using 10,000 problems, with sizes selected in the range from n=7 to n=65. These problems are generated randomly (again by using Potts and Van Wassenhove's method [51]) to produce a uniform sample of patterns across the problem domain. The patterns are to be classified such that ten categories are described. The number ten is chosen arbitrarily. The greater is this number, the more specialized are the sequencing networks which, intuitively, may lead to a better overall performance. The tradeoff, of course, is that more training time is needed and a larger system has to be managed. MTCLASS has a 4-unit input layer that accepts continuous-valued input data vectors. Each vector for a n-job problem is described by :

P1 = minimum due date;

P2 = average due date;

P3 = maximum due date; and

P4 = 100.00 (a reference value).

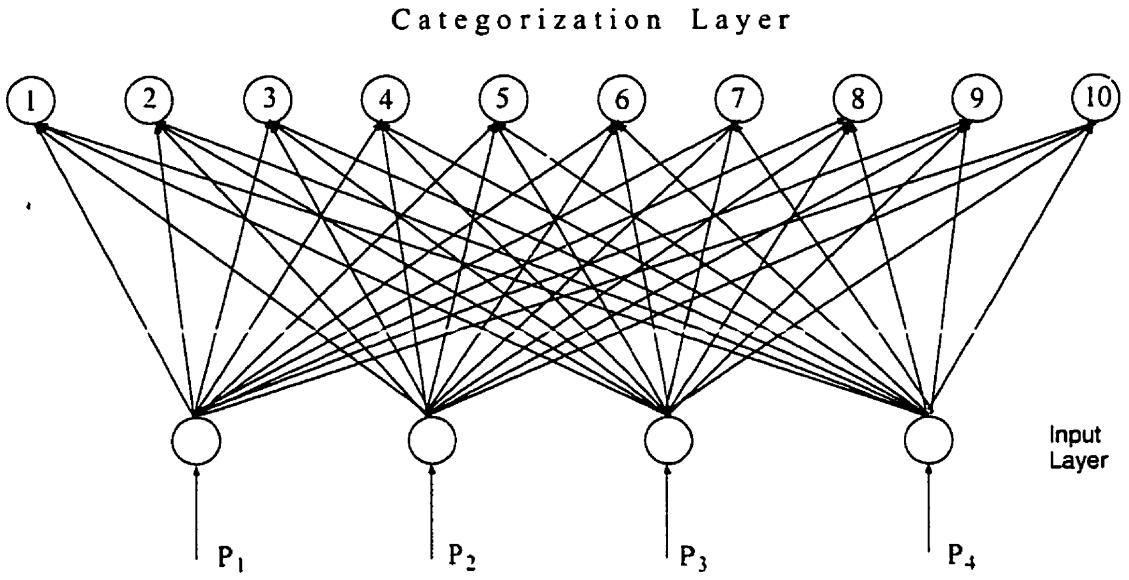


Figure 4.7: A 10-category classifier.

The first three variables are selected because they are considered the 'defining' characteristics of a problem in the domain under consideration. P4, the reference value, is used as a reference that effectively normalizes the data within the domain. The trained MTCLASS network is able to classify unknown problems by taking a vector (P1, P2, P3, P4) at its input layer and, after performing what amounts to a least-squares error calculation, selecting the output unit that represents the category that the input vector best fits. Training the Classifier Network is performed by using the DESIRE/NEUNET software again. Figure 4.7 shows the structure of a competitive network for classifying the vectors into ten categories. Details of the training and testing of MTCLASS are available in section A.4 of Appendix A.

The neural sequencers in the NJCASS are trained separately, one network for each of the ten categories, by using data generated only from problems falling under

the same category. In order to satisfactorily train each one of these networks, considerable training and experimentation is involved with different numbers of units in the hidden layer. The effort and time for training can be significantly reduced by training the neural sequencer networks only up to a point at which the average positioning error, e_q, is about 1.5. This value implies that the jobs in the resulting sequences may be expected to deviate from their correct locations in the optimal sequence by one or two positions. Such sequences may be improved further by a post-processing stage involving the interchange of positions between adjacent jobs.

4.5.1 Post-processing

All sequences produced by the neural sequencers are subjected to post-processing that uses an adjacent pairwise interchange strategy. The interchange strategy moves from left to right across a sequence and evaluates, at each position, the effect of an interchange between the job currently occupying that position and the job in the immediately following position. An interchange is preserved when it leads to an improvement in the value of the objective function. Otherwise, the interchange is nullified. When the last position in the sequence is reached, the procedure is repeated, starting from the first position, until no further interchanges can be made in one complete pass through the sequence. This interchange strategy is effectively a descent mechanism that forces the current solution to a local minimum. Due to the unidirectional movement of the interchanges, there always exists the possibility that jobs already located in optimal locations may be pulled out of position. The implication is that an initial sequence in which many jobs are already located in their desired optimal positions is likely to be perturbed quickly into a local mini-

mum that is very close to the initial sequence. Therefore, the initial sequence in this instance needs to have the maximum number of jobs located in their optimal positions. On the other hand, a starting solution where more jobs are near their desired positions (i.e. out of place by only one or two positions) appeals more to the particular interchange strategy used in the post processing stage. With a larger number of jobs tolerated out of position in the starting sequence, the need to train the networks to a very low error, e_q, is reduced substantially. This reduction translates into less training time for these networks.

4.5.2 NJCASS for Minimizing the Mean Tardiness

Ten neural sequencers are trained for the NJCASS to minimize the mean tardiness. The training procedure for each category is similar to that employed for the previous networks, but with differences regarding when to accept the network as satisfactorily trained. The training sets are derived from 12-job example problems that are solved by dynamic programming. For the neural sequencers specialized for this NJCASS, the training is stopped after every 100 cycles and the network is tested by employing a test data set. During this testing, the sequence generated by the neural networks is post-processed by the interchange strategy. The training is stopped when the test results appear to have 'bottomed out'. This occurs at between 300 and 5000 training cycles, depending on the category. In three of the ten categories, it is observed that the post processing interchange strategy results in near optimal solutions when applied to initial sequences sorted in the order of their non-decreasing due dates (EDD). It is decided, therefore, to train the neural sequencers for these three particular categories to sequence the jobs in the order of

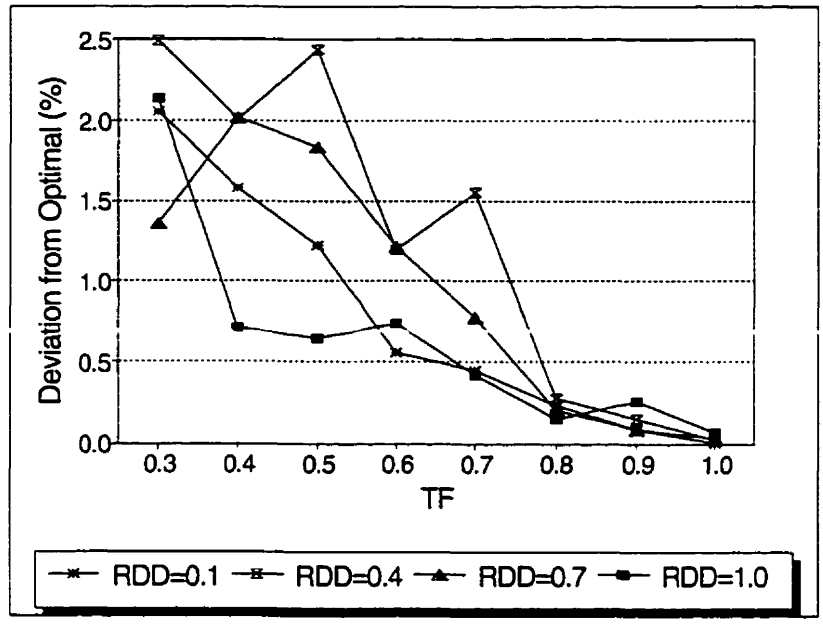


Figure 4.8: Minimizing the mean tardiness for 12-job problems with NJCASS.

the earliest due dates, rather than the optimal sequences. Section A.5 of Appendix A contains details for the training of each of these sequencers.

The trained and completed NJCASS is tested for different 12-job problem sets. The test data contains 20 sets of 200 problems, each set generated with specific TF and RDD values covering the problem domain. The results, plotted in Figure 4.8, show an average deviation from the optimal of less than 1% for TF values of 0.7 and above. The deviation is slightly higher for problems generated with a TF of less than 0.6 (i.e. problems with looser due dates). The critical question is: how well does this NJCASS, which is trained on the basis of information from 12-job examples, perform when faced with larger sized problems? To examine this issue, a comparison is undertaken by using the adjacent pairwise interchange (API) heuristic of Fry

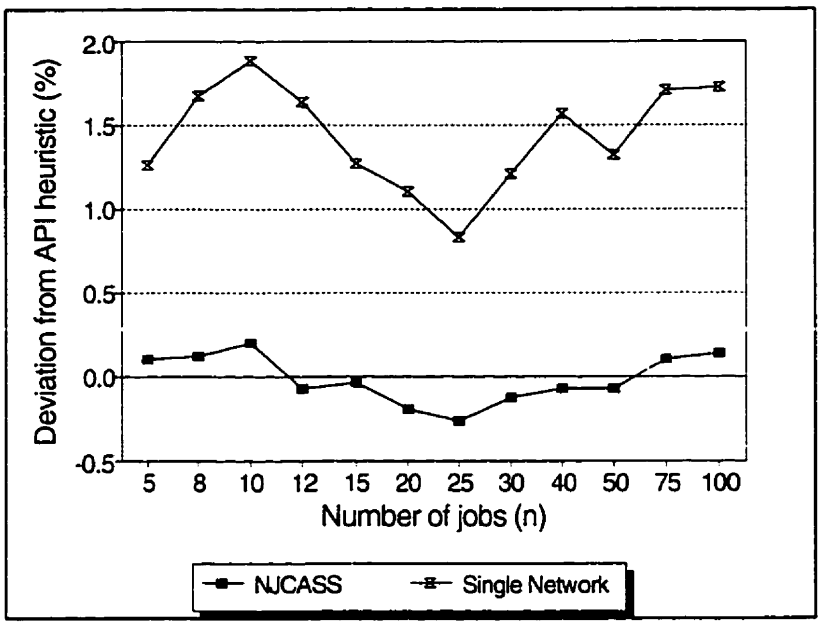


Figure 4.9: NJCASS Performance with increasing job size, n.

et al. [71] with the modification proposed by Koulamas [72]. This heuristic applies three different interchange strategies to initial sequences generated by using three different sorting rules. The best outcome from the nine possible combinations of initial sequence and interchange strategy is selected. The heuristic's main drawback is that long execution times are required, but the results are reported by Koulamas [72] to be good for problems having up to 100 jobs.

The performance of the NJCASS, plotted in terms of the percentage deviation from the results of the API heuristic, is displayed in Figure 4.9. Each test set contains 200 problems that are generated randomly across the domain. It can be seen from the figure that the NJCASS performs better than API in the cases involving problem sizes between 10 and 50 jobs. For more than 50 jobs, the deviation in

performance between the two approaches is in favour of the API. The reason for the gradual deterioration observed in the performance of NJCASS, starting from about n=25, is because it is trained based on target values which are derived from merely 12-job problems. This approach is apparently not sufficiently accurate for more than 50 jobs. The loss in accuracy, as n grows, is a result of the current positioning error becoming relatively weaker. In other words, a positioning error encompasses more positions as n grows because increasingly more jobs have to fit within the interval between G_i and G_{i+1}. The result is that a job may be off-target, on average, by several positions rather than one or two for a given positioning error of, say 1.5, that is based on 12-job training problems. Then simple adjacent interchanges have greater difficulty in bringing an off-target job to the desirable sequential position. The remedy is to improve the target accuracy by basing the training on data from problems in which n is greater than twelve, where possible. The conclusion from Figure 4.9, on the other hand, is that the NJCASS can compete, for up to 50-job problems, with the API heuristic.

The API heuristic based on Fry et al. [71] and modified by Koulamas [72] is chosen in this comparison for two reasons. First, its solution is competitive with the better heuristics for minimizing the mean tardiness, as reported in Table 1 of ref. [72]. Second, the API is a general purpose interchange heuristic, the application of which need not be limited to one specific performance criterion. Figure 4.9 demonstrates that the result of an initial sequence produced by the neural network and modified by a simple adjacent pairwise interchange is comparable, if not slightly better, than that generated by the nine different combinations of initial sequence and interchange strategies encompassed in the API heuristic.

As a further evaluation of the solution quality of the NJCASS, a comparison is made with one of the several leading heuristics that are specific to the mean tardiness criterion. For example, Russell and Holsenback [73] demonstrated that the PSK heuristic of Panwalker, Smith and Koulamas [58] and the Net Benefit of Relocation (NBR) heuristic of Holsenback and Russell [56] are more or less comparable. The NBR is stronger in certain cases, while the PSK is better in other cases. For convenience, the NJCASS is compared next with one of these heuristics, namely the NBR algorithm taken from ref. [56]. The latter uses a dominance rule, based on Emmons' conditions of optimality [57], together with calculations to determine the net benefit of job relocations in the sequences. The comparison is performed by using a procedure identical to that employed in ref. [56]. Twenty sets of test data are used: each set contains ten problems of n jobs where n is 12, 25 or 50. Individual sets are generated by utilizing the Potts and Van Wassenhove method [51] for specific TF and RDD values. The 12-job problems are tested because that is the problem size used in the training of the neural sequencer networks. The tests for the 25 and 50 job problems are done to evaluate how relatively well the NJCASS performs for larger problems. Table 4.6 presents the results. The integers under the columns headed 'Best' indicate the numbers of times NJCASS achieved a solution better than or equal to NBR in each set.

Table 4.6 shows that NJCASS produces solutions which are generally comparable to those given by NBR. The NJCASS is better in 26 out of the 60 sets tested. NBR is better in 24 sets, and the two methods give equal total tardiness in the remaining sets. On the other hand, NBR is slightly superior based on the total

Table 4.6: Total tardiness for test sets solved by NJCASS and NBR.

		$n = 12$			$n = 25$			$n = 50$		
TF	RDD	NJCASS	NBR	Best*	NJCASS	NBR	Best	NJCASS	NBR	Best
0.2	0.2	753	724	8	2213	2057	4	7131	6577	1
0.2	0.4	137	123	7	451	322	5	597	309	2
0.2	0.6	15	15	10	0	0	10	0	0	10
0.2	0.8	37	37	10	0	0	10	0	0	10
0.2	1.0	89	89	10	0	0	10	0	0	10
0.4	0.2	3460	3406	7	12610	12632	5	47802	47962	4
0.4	0.4	2460	2433	6	8361	8377	5	31647	31793	4
0.4	0.6	1721	1725	9	4773	4590	7	15810	16050	4
0.4	0.8	1092	1113	10	2017	2017	10	4649	5206	10
0.4	1.0	1174	1133	9	1584	1575	8	2340	2424	10
0.6	0.2	8275	8296	5	32322	31885	3	125873	124524	1
0.6	0.4	7403	7433	6	29075	28270	0	113553	111365	2
0.6	0.6	6958	6905	7	25717	25135	3	98213	98532	5
0.6	0.8	6610	6642	8	23833	23840	6	93621	94140	5
0.6	1.0	7075	7102	9	24550	25004	9	100676	102501	8
0.8	0.2	15421	15400	6	61503	61332	4	242122	241396	2
0.8	0.4	15245	15232	8	60151	59979	4	239450	238481	1
0.8	0.6	15637	15638	10	61398	61195	3	246925	247438	8
0.8	0.8	16232	16257	9	63487	63653	4	259486	260069	7
0.8	1.0	16850	16857	10	66042	66296	9	273479	274453	9
Total		126664	126560		480087	478159		1903374	1903220	

* Number of times NJCASS solution is better than or equal to the NBR solution.

tardiness values summed over the 20 data sets for each problem size n. This result is due mainly to a particular weakness of NJCASS in those data sets having low RDD. It is not surprising because the optimal sequences in such cases exhibit no strong patterns or rules that help the neural networks to learn the relationships between the input data and the desired sequence. Nevertheless, NJCASS can be enhanced then by increasing the number of categories covering this region of the problem. The improvement to be expected is reflected in the behaviour illustrated in Figure 4.9. This figure shows what happens when merely a single neural network is trained to cover the entire problem domain rather than the 10 currently used by NJCASS. The overall trend from the single neural sequencer is similar to that from the NJCASS, except that it is about 2% less accurate. This deterioration reduces the accuracy to a level that renders the single network's performance marginal in comparison to that of heuristics such as those mentioned in ref. [72].

4.5.3 A Limited Exponential Cost Function

A hypothetical situation involving penalty costs that exhibit a limited exponential behaviour is described now. The example is used to illustrate how the NJCASS can be implemented in situations where no heuristic is known beforehand for optimizing the given criterion. The mean tardiness measure applies a penalty that increases in linear proportion to the tardiness. In extreme cases, it is possible that the penalty may actually exceed the value of the job. The cost function considered next differs in that a substantial proportion of a total penalty is assessed at the instant of tardiness. As the tardiness increases, the penalty follows an exponentially decreasing rate up to a maximum value. This limited exponential cost function is

expressed in the following form:

$$Z = \sum_{i=1}^{n} \frac{100 \times t_i}{1 + e^{-0.05T_i(S)}} \quad . \tag{4.4}$$

Two variants of this function are analyzed and tested with a NJCASS set up for the purpose.

Case (i)

The first scenario has n jobs due at different times. The tardiness penalty per time unit is uniform for all jobs (i.e. $t_i = 1 \; \forall$ i). It is applied in the limited exponentially increasing fashion of Equation (4.4).

Case (ii)

The tardiness penalty is not uniform but differs from job to job. The training and test data for the NJCASS has t_i selected randomly from the uniform distribution [1,10].

The NJCASS for minimizing cost function (4.4) also has ten categories, the number of categories are selected arbitrarily again for illustration purposes. It is trained and constructed in a manner identical to that discussed previously for minimizing the mean tardiness. Presuming that no quick method is available for minimizing cost function (4.4) in large sized problems, complete enumeration is used to determine the solutions needed for the training problems. Thus, the sequencer networks are trained for each category by using 8-job example problems.

Table 4.7: Generation parameters for randomly selected problems.

	Set A	Set B
TF	0.7	0.8
RDD	0.5	0.8

The hypothetical cost function of equation (4.4) is tested for cases (i) and (ii) by using the NJCASS. Two randomly found data sets, each containing 200 problems, are generated from the combinations of TF and RDD given in Table 4.7. These combinations are selected as representative of cases involving a moderate tightness and range of due dates. Finding a good heuristic for minimizing Equation (4.4) may range from using 'quick and dirty' methods to more involved techniques. The API heuristic is chosen for comparison because, not only does it combine several of the prominent 'quick and dirty' rules, but it further enhances their results by using three different interchange strategies. The API results, when taken in the context that they come from an approach that is based on a rule-of-thumb, can be considered reasonably good, even for a function like that used in Equation (4.4). The comparison between the API and NJCASS approaches is displayed in Figure 4.10.

The results in Figure 4.10 are presented in terms of the percentage deviation of the NJCASS results from those of the API heuristic. A negative deviation means that the API heuristic is worse. Therefore, the general superiority of the NJCASS over the API approach is demonstrated clearly in Figure 4.10 for the case (i) and case(ii) scenarios.

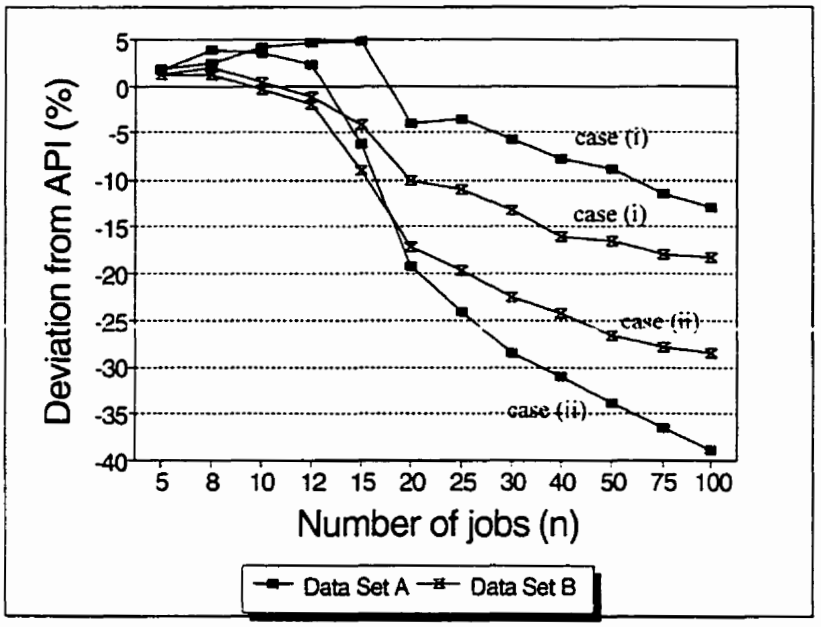


Figure 4.10: Performance of NJCASS for limited exponential function.

4.6 Conclusions

Based on the tests performed for a limited number of performance criteria, the NJCASS appears to be a highly competitive procedure for sequencing jobs on a single machine. Its performance may be improved even further by increasing the number of categories or by experimenting with other categorization schemes. One possibility is to train a neural network for problems that represent the desired domain and then isolate those problems that can be solved optimally by the network. Then, the remaining problems are used as a source for training another network, and so on until all the training problems are covered by individual, specialized networks.

The neural network approach highlighted by the NJCASS provides several advantages, but chiefly the flexibility of application for different performance criteria. Special purpose heuristics, developed for a specific criterion, may be unsuitable when a different criterion is considered. Furthermore, a heuristic may perform well, on average, but still remain consistently weak for a certain class of problems. The NJCASS and its classification scheme allow what are effectively customized 'heuristics' for problem classes having similar characteristics within a domain. The user needs to identify only key characteristics and the desired number of categories. The unsupervised learning algorithm results in a Classifier Network that is capable of suggesting a suitable categorization based on information from the vector data with which it is trained. Thus, the procedure can be conceivably automated with the categories self-generated through unsupervised learning. Randomly generated problems can be produced to represent these categories and the problems can also be used subsequently to train the specialized sequencer networks.

Any neural network approach possesses the disadvantages of neural networks in general, namely the time to train and test the networks as well as the experience required to achieve good results. Although the NJCASS's performance is quite competitive, it is unlikely to be noticeably better than a good heuristic developed painstakingly for the purpose. Yet, in the event that no such heuristic is available, the NJCASS appears to be a better alternative to makeshift measures based on intuition or modification of existing rules or heuristic procedures. Moreover, the development time for NJCASS is much shorter than that needed to evolve a full-blown heuristic procedure. It should be noted, however, that neural sequencing appears best suited to performance measures that are regular. A regular perfor-

mance measure is one where the value of the function can grow only if at least one of the completion times in the schedule increases [8]. Initial results using neural sequencing for a criterion involving an irregular performance measure, namely minimizing the completion time variance (Merten and Muller [74]), suggest that it is not much more effective than simple sorting heuristics. However, further research is still needed.

Chapter 5

Dynamic Scheduling with Cooperative Dispatching

5.1 Introduction

Scheduling in manufacturing cells is frequently dynamic in nature. The dynamism arises primarily under two circumstances. The first is when jobs arrive continuously at a cell and join other jobs which are already waiting in queue. This differs from the static case, where a fixed number of jobs is available at the start of the scheduling and the initial schedule remains valid until all the jobs are completed. The second circumstance arises when unexpected or unpredictable events, such as a machine failure or a defective product, invalidate the current schedule.

Scheduling under dynamic conditions implies the need to continually update the schedule. The terms 'on-line scheduling', 'real-time scheduling' and 'dynamic scheduling' are often used interchangeably to describe situations requiring a continuous schedule modification. In fact, there is a fine distinction between real-time and on-line scheduling. Real-time scheduling may be performed either on-line or off-line. If it is performed off-line, then a "snapshot" of the cell's current status and activities is captured and a new schedule is devised almost instantaneously. The need to efficiently finalize this 're-scheduling' in a matter of seconds poses computational difficulties, particularly if there are many jobs waiting. If on-line scheduling is employed, on the other hand, the scheduling is reduced to making dispatching

decisions whenever a resource becomes available. This reduction in the scope of the scheduling problem to a series of 'as-needed' decisions makes on-line scheduling attractive for real-time scheduling demands.

The CD approach described in Chapter 3 can be performed on-line, like other dispatching rules. Unlike most simple dispatching rules, however, CD is more complex and requires real-time information regarding the current states of the jobs and machines in a cell. Current computer networking and cell control technology, however, allow this information to be collected automatically so that the CD approach is implementable for an FMC. The remainder of this chapter describes the performance of CD in dynamic scheduling. The evaluation of CD is undertaken by using a simulation approach for dynamic job arrivals.

5.2 CD Scheduling with Dynamic Job Arrivals

FMCs often have to process orders that arrive randomly. The unpredictability of the time of arrival poses obvious problems because existing schedules, if any, become obsolete immediately. On-line scheduling treats a new arrival simply as an additional job joining a queue. The next time that a machine needs to be loaded, the new arrivals are considered immediately, together with the previous jobs waiting to be dispatched. Therefore, with on-line scheduling based on dispatching rules, no major scheduling disruption is experienced as a consequence of the new arrivals.

5.3 Simulation

The performance of CD is evaluated experimentally for dynamic job arrivals. The experiments are performed by means of a computer program written in the C language [75]. They are designed to simulate activities related to the movement of parts through a FMC. The program monitors each job's starting and finishing times on the machines, from which the desired performance data is collected. The program allows simulations having a selected number of dispatching rules, in addition to CD.

The objective of the simulations is to study the comparative effectiveness of CD for different job arrival rates. The jobs are assumed to be arriving at the cell according to a Poisson process. In this process, the expected number of arrivals occurring in an interval of time, t, has a Poisson distribution with a parameter $\lambda_q t$ in which λ_q is the expected number of arrivals per unit time. Assuming that the number of arrivals in each non-overlapping interval are independent of one another, then the time between successive arrivals is distributed exponentially with parameter λ_q. The higher is λ_q, the greater is the frequency of arrivals and the more severe is the likely congestion. The evaluation of CD takes the form of a comparison with the traditional dispatching rules that are expected to be the most effective for the given scheduling criterion. The individual criteria that are considered and tested, under different job arrival rates, are the minimum mean flowtime and the minimum mean tardiness.

5.3.1 Generation of Test Data

The data for each simulated test problem relates to 500 jobs. Initially, two jobs are assumed to be already in the system and awaiting their first operations. The arrival times for these two jobs correspond to time zero. Each new subsequent job arrives at an instant that is generated randomly. Arrivals occur at intervals that are distributed exponentially. The length of the interval between arrivals, t_r, is determined randomly from:

$$t_r = \frac{ln(p)}{-\lambda_q} \tag{5.1}$$

where p is a real-valued random number from the uniform distribution U[0,1]. The arrival time of each new job is determined by incrementing the previous job's arrival time by t_r. This procedure is repeated until 500 job arrivals have been generated. Each test set contains 20 different problems that are created randomly in this fashion for a specific arrival rate, λ_q, and a given performance criterion.

In order to study CD's performance in minimizing the mean tardiness, the method of Potts and Van Wassenhove [51], which was employed in the previous chapter, is adapted for generating problems having dynamic arrivals to multiple-machine cells. The main modification is in computing a value for P, which is the main influence on the value of the generated due date (see section 4.3.1). In the static case, P is simply the sum of the processing times for all the jobs in the problem set. Now P is an estimate based on the expected flowtime that a job would have assuming FIFO processing, plus the sum of the processing times on the individual machines for that job. The value of P that is computed for the i^{th} job is then used in Potts and Van Wassenhove's method [51] to generate a due date for that job.

5.3.2 Minimizing the Mean Flowtime

Experiments are performed to minimize the mean flowtime for the three configurations shown previously in Figure 3.1. Processing times for each job on each of the four machines are taken as random integers from the uniform distribution U[1,35]. Separate tests are performed by using arrival rates, λ_q, of 0.030, 0.035, 0.040, 0.045 and 0.050. Arrival rates lower than 0.030 are not considered because the traffic generated is too sparse to be meaningful. In these situations, every dispatching policy produces nearly identical results which approach those of FIFO. On the other hand, arrival rates higher than 0.050 are near the situation when the rate of arrival is greater than the cell's processing capacity (i.e. the service rate). Then, the queue of waiting jobs increases perpetually. Hence, the arrival rates of interest for the range of processing times and configurations under consideration are those between 0.030 and 0.050. Each data set is replicated by utilizing the SPT, LWKR and FIFO dispatching rules, in addition to CD. The former rules are selected for the same reason they were used in the static problems of Chapter 3.

The results of the simulations are presented in Figures 5.1 and 5.2. In Figure 5.1, all the intermediate buffers of each of the three configurations are constrained to a FIFO selection from the waiting jobs. This FIFO constraint is relaxed in Figure 5.2. The results given in Figures 5.1 and 5.2 are expressed, like those presented in Chapter 3, in terms of the mean performance ratio (PR) evaluated over each set of 20 problems. This ratio uses the mean flowtimes produced by the FIFO dispatching rule as the reference. The PR for another method, say method A, is obtained by

dividing the mean result from the reference method by that from method A. A PR value that is greater than 1.0 is a measure of the percentage superiority of method A's results over those obtained by the reference method.

It can be observed from Figure 5.1 that CD outperforms the SPT and LWKR rules by as much as 5% in the Type I and Type III configurations, but only by about 1% in the Type II configuration. The general trend apparent in Figure 5.1 is that CD performs increasingly better than these two rules as the arrival rate grows. This behavior indicates that CD is more effective in minimizing the mean flowtime under conditions of high cell congestion. The trends witnessed in Figure 5.1 appear to hold also for the cases shown in Figure 5.2, i.e. for non-FIFO intermediate buffers. Again, CD increasingly outperforms the other two dispatching rules as the arrival rate grows, albeit at a magnitude not exceeding 2%.

The closeness in performance to FIFO that is observed in Figure 5.1 for all the dispatching rules in the Type II configuration is because the job arrivals are shared initially between two machines rather than only one machine. With an arrival rate of 0.05, the average queue lengths for the first two machines rarely exceed 2 or 3 jobs in the Type II configuration. The average queue at the third machine, which receives jobs from both the first two machines, is significantly higher. When the queue at the buffer for the third machine, which is a strong bottleneck, is constrained to FIFO processing, all the other dispatching rules are not significantly more effective than FIFO. This is evidenced in Figure 5.1. On the other hand, if the bottleneck is not constrained to a FIFO selection, then the dispatching rules show improved results in comparison to FIFO. See Figure 5.2.

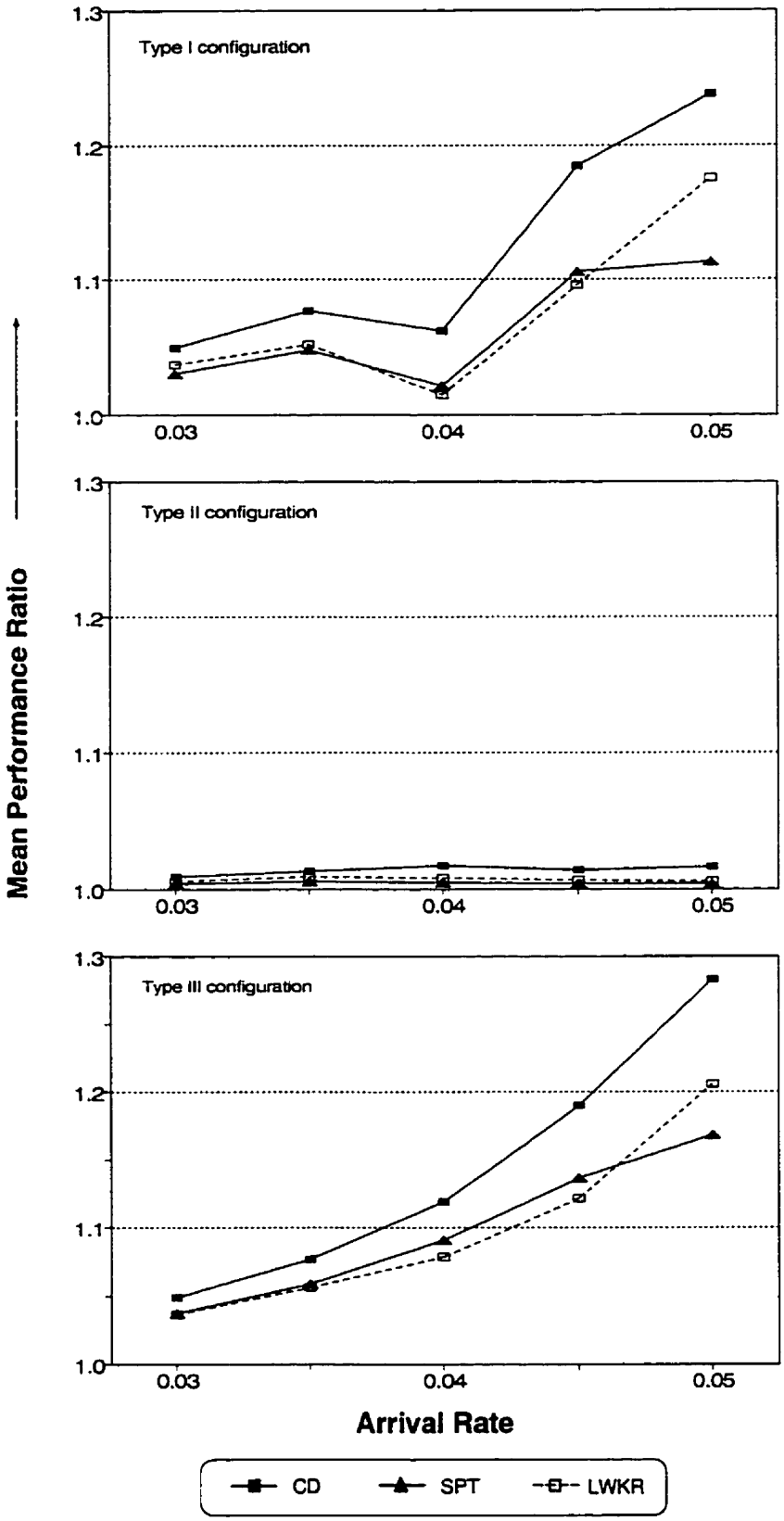


Figure 5.1: Minimizing the mean flowtime with FIFO intermediate buffers.

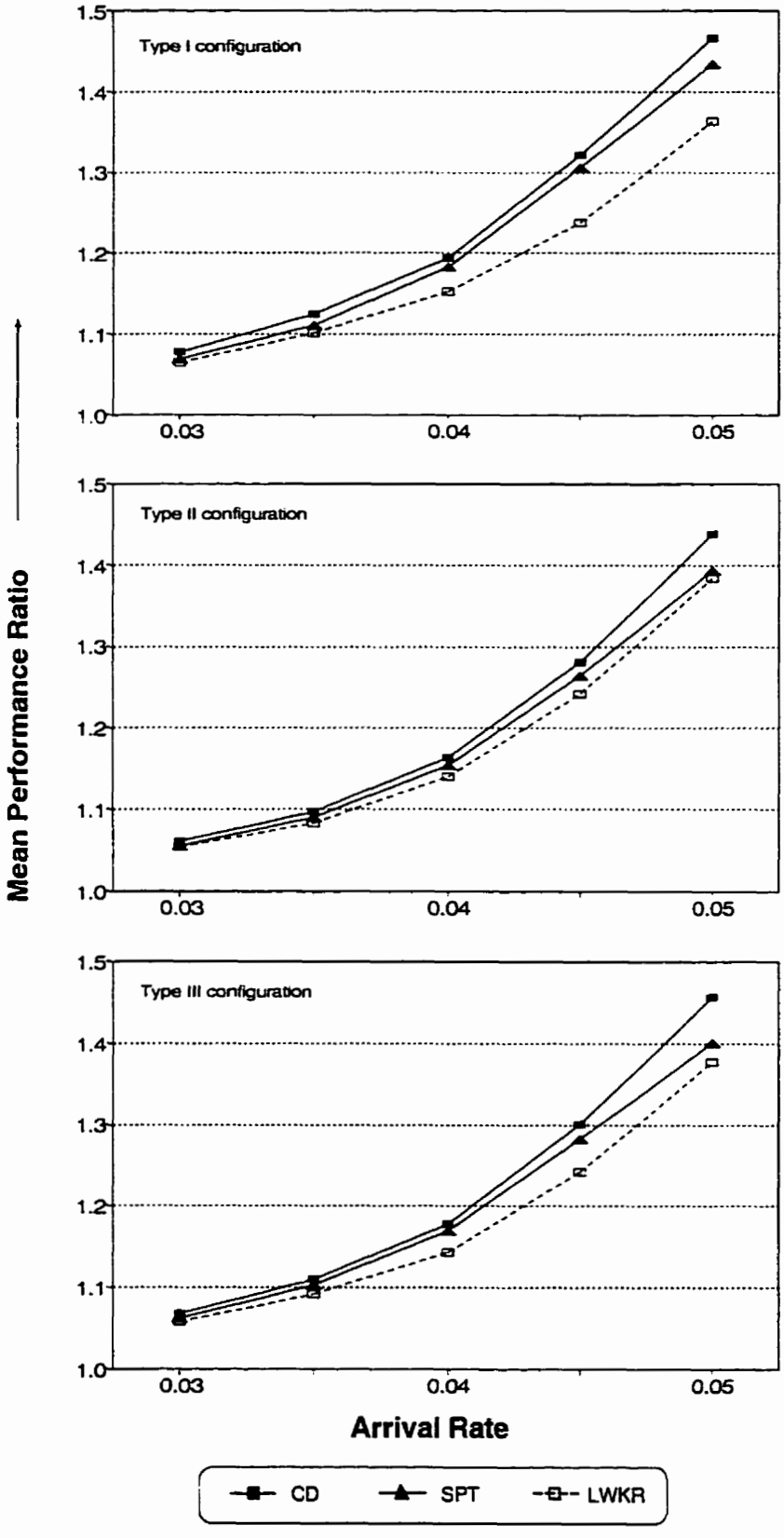


Figure 5.2: Minimizing the mean flowtime with non-FIFO intermediate buffers.

A measure of the effectiveness of each dispatching rule used is the number of times this dispatching rule provides the best solution in the test data sets. This statistic is presented in Table 5.1 for the four dispatching rules employed in the mean flowtime minimization experiments with FIFO intermediate buffers. The values given in Table 5.1 combine the simulation results from the different arrival rates considered for each of the three configurations. The table indicates that CD gives the best result in all the test problems for the Type I configuration, in about 83% of the problems involving the Type II configuration, and around 93% of the problems for the Type III configuration. The significance of these percentages is that, although CD appears from Figure 5.1 to be only marginally better than the other dispatching rules, it is actually the policy that is better most frequently.

Table 5.1: Best solution frequency (in percentage) for mean flowtime and FIFO intermediate buffers.

Configuration	CD	SPT	LWKR	FIFO
Type I	100.00	0.00	0.00	0.00
Type II	83.00	7.00	9.00	1.00
Type III	93.00	7.00	0.00	0.00

A similar analysis is presented in Table 5.2 for the simulations involving non-FIFO buffers because Figure 5.2 shows that CD and the other dispatching rules perform closely, when measured in terms of average results over the test problems. However, Table 5.2 indicates that CD outperforms these dispatching rules in about 86%, 84%, and 86% of the problems tested for the Type I, Type II and Type III configurations, respectively.

Table 5.2: Best solution frequency (in percentage) for mean flowtime and Non-FIFO intermediate buffers.

Configuration	CD	SPT	LWKR	FIFO
Type I	86.00	14.00	0.00	0.00
Type II	84.00	7.00	9.00	0.00
Type III	86.00	14.00	0.00	0.00

The CD dispatching rule is tested next for minimizing the mean flowtime in a pure flow shop having six machines. The same simulations are conducted for this Type I configuration in order to evaluate the consistency of CD's performance for cells having more than the previously considered number of machines, namely three. The results are displayed in Figure 5.3 for both cases where the buffers are constrained or unconstrained to a FIFO selection. They indicate that, when the intermediate buffers are constrained to the FIFO selection, CD is consistently superior to the SPT and LWKR rules and, again, increasingly so as the arrival rate grows. However, when all the intermediate buffers are not constrained, CD's performance is comparable to those of the other two dispatching rules for arrival rates of 0.040 and less. At higher rates of arrival, SPT starts to perform better, reaching approximately 2% superiority over CD for an arrival rate of 0.050.

It is apparent, at least in the pure flowshop, that as more machines are added to a cell, the more critical becomes the influence of FIFO versus non-FIFO processing in the intermediate buffers. CD outperforms the other dispatching rules by a wider margin as the number of buffers that are constrained to FIFO processing increases.

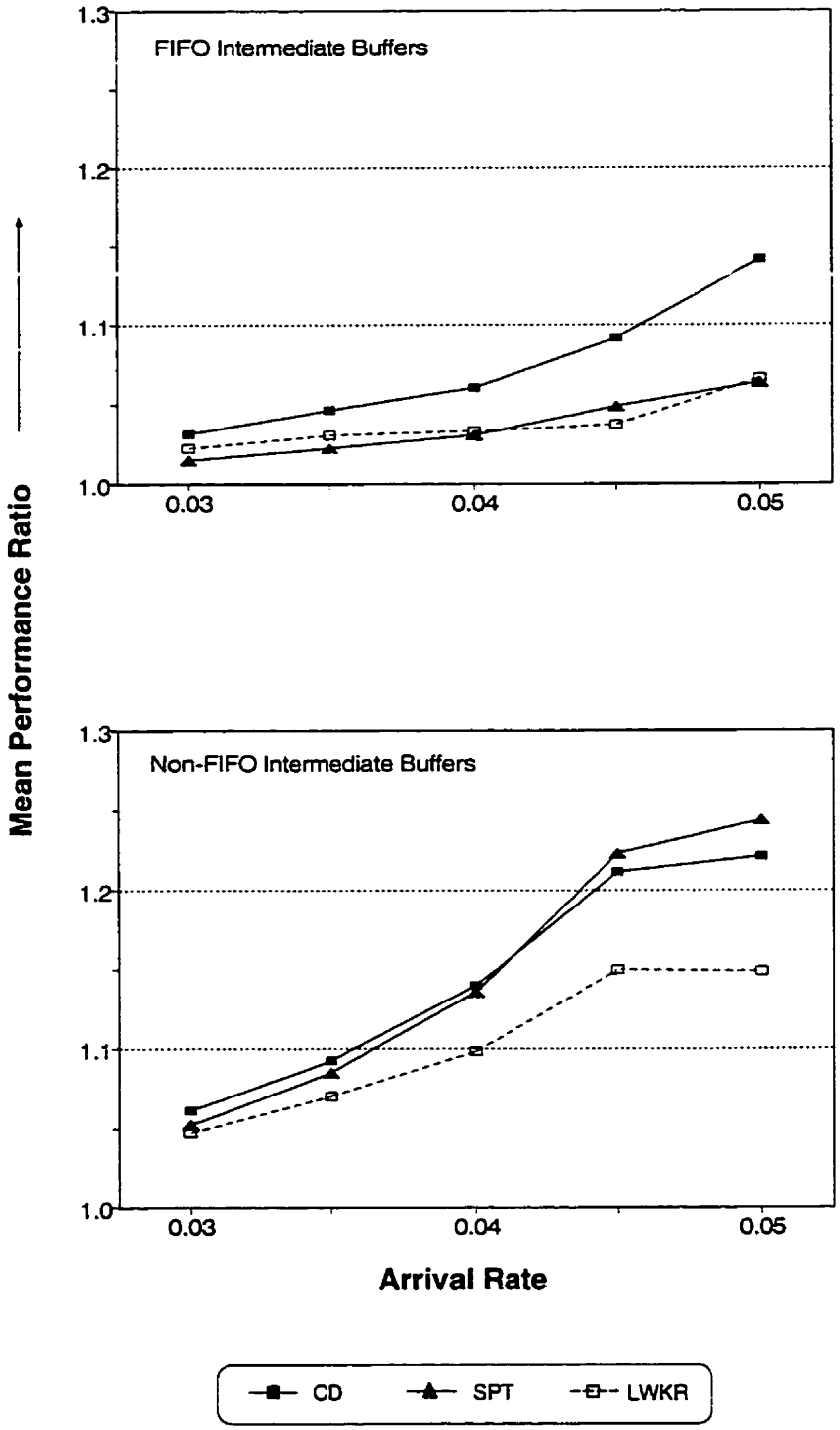


Figure 5.3: Minimizing the mean flowtime in a Type I configuration with 6 machines.

On the other hand, the other dispatching rules are at their best when the number of constrained buffers, particularly in bottleneck machines, is minimum. A comparison of the results presented in Figure 5.3 with those shown in Figures 5.1 and 5.2 indicates that CD's power tends to diminish as the number of machines grows in a cell. Despite this trend, CD remains competitive in problems for which the other dispatching rules excel, such as cases involving non-FIFO buffers and high arrival rates.

5.3.3 Minimizing the Mean Tardiness

The evaluation of CD for minimizing the mean tardiness is conducted next by employing simulation experiments on data sets organized as follows.

1. Each data set contains 20 randomly generated problems; each problem covers 500 job arrivals.

2. Data sets are generated for the arrival rates of 0.030, 0.040 and 0.050.

3. The due date for each job is specified by using the modified Potts and Van Wassenhove method with TF and RDD equal to 0.4 or 0.8, respectively. Thus, there are four possible combinations of these TF and RDD values: TF=RDD=0.4; TF=0.4 & RDD=0.8; TF=0.8 & RDD=0.4; and TF=RDD=0.8.

The simulations are undertaken in a manner similar to that used for the mean flowtime criterion. However, in this instance, the dispatching rules to which CD is compared are : 1) SPT; 2) EDD; 3) MDD; 4) minimum remaining slack (MSLK); and 5) FIFO. With the exception of FIFO and SPT, these dispatching rules are selected because they are due date based rules. Due date based rules are a logical choice for

satisfying due date based criteria of which the minimum mean tardiness is one. SPT is included because of its effectiveness when many of the jobs are inevitably tardy [8].

5.3.3.1 Cells with FIFO Intermediate Buffers

Simulations are performed for the three configurations shown in Figure 3.1. They are done first for FIFO-constrained intermediate buffers and subsequently for non-FIFO buffers. Figures 5.4 through 5.6 show the results for the former case. The results for a pure flowshop having three machines (i.e. a Type I configuration) are presented in Figure 5.4. They are expressed again in terms of the mean performance ratio but, this time, the references are the mean tardiness results obtained with the FIFO dispatching rule. Each of the curves shown in Figure 5.4 corresponds to one of the three different arrival rates and one of the four combinations of TF and RDD values considered. Results for similar data for the Type II and Type III configurations are given in Figures 5.5 and 5.6. Figures 5.4 through 5.6 reveal a consistent superiority of CD over the better of the other dispatching rules. The following general observations can be made.

1. The overall trends are similar for all three configurations.

2. The difference in the performance superiority of CD, relative to the best of the other dispatching rules, widens yet again with increasing λ_q.

3. CD's superiority over the next best dispatching rule is greatest (by approximately 25%) for the data sets generated with a high λ_q and low TF and RDD values.

4. The use of non-FIFO buffers for severe bottleneck machines, as seen in the Type II configuration, greatly improves the performance in the cell.

Table 5.3 compares the frequency, expressed as a percentage of the total number of problems tested, with which each dispatching rule provides the best solution in the test problems. The results are sorted according to the configuration and the due date tardiness factor, TF. Table 5.3 indicates that CD dominates in the Type I and Type III configurations. It is also the better in more than two out of every three problems for the Type II configuration.

Figures 5.4 through 5.6 generally show a steady decline in the performance ratio as TF increases. This trend is attributable to the increasing total tardiness accompanying high TFs. When the total tardiness numbers are low, small differences are magnified if the numbers are expressed as a percentage of one of the tardiness values. The converse is true for large tardiness numbers. Hence, a further analysis of the results is undertaken by identifying the number of times that each dispatching rule gives the best solution. It is performed by using the Relative Deviation Index (RDI) proposed by Kim [46]. The RDI measures the deviation from the best and worst results obtained by the methods being compared. The RDI is defined as $(T_a - T_b)/(T_w - T_b)$. T_a is the result given by the method under evaluation. T_b and T_w are the results given by the best and worst solutions, respectively.

Table 5.4 combines the results for all tardiness factors in each configuration. They are expressed in terms of the mean RDI and the corresponding standard deviation. A mean RDI that is nearer zero indicates a greater consistency in giving the best solution. A mean RDI around 1.0 implies that a particular method tends

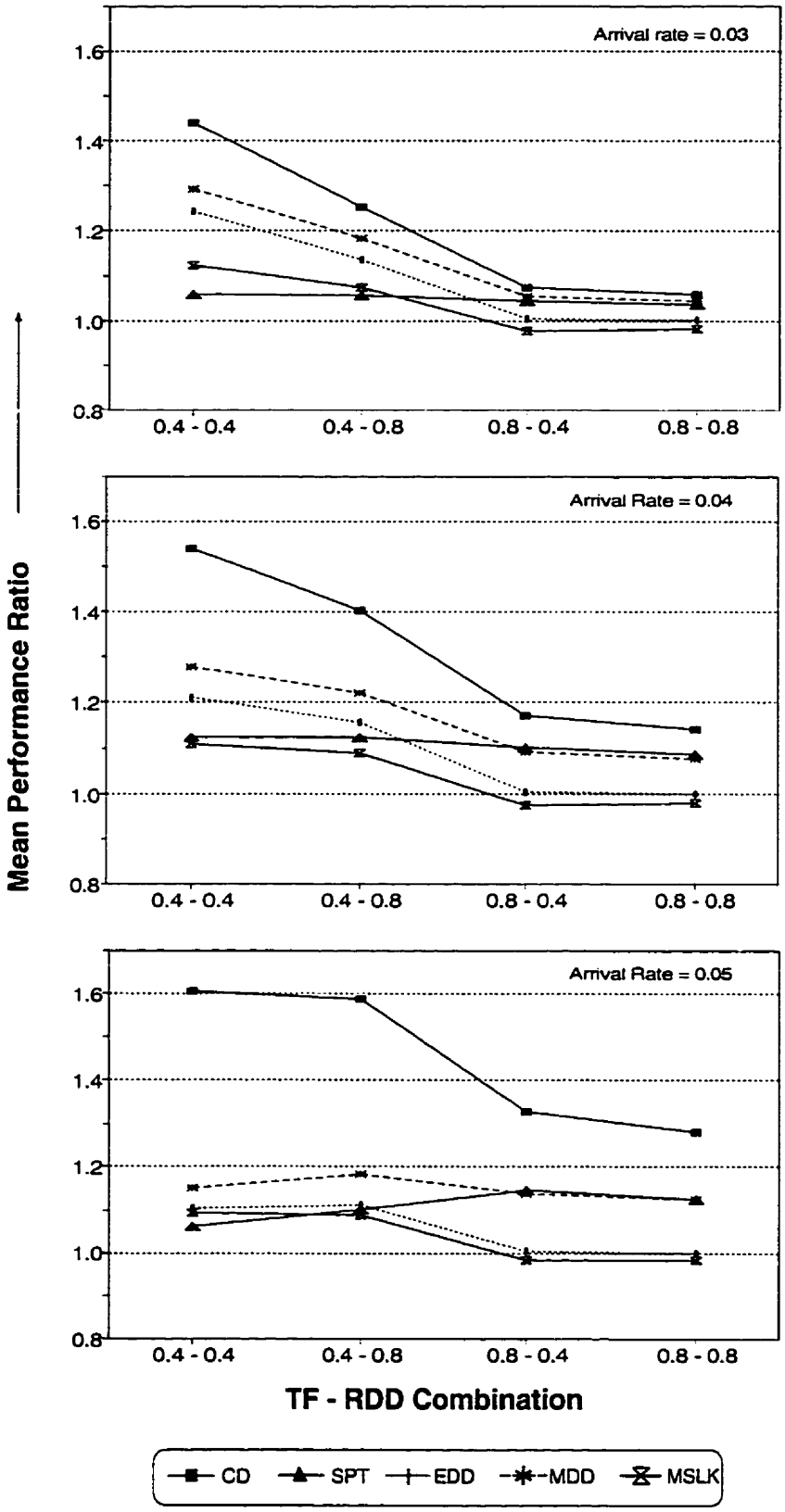


Figure 5.4: Mean tardiness for Type I configuration with FIFO buffers.

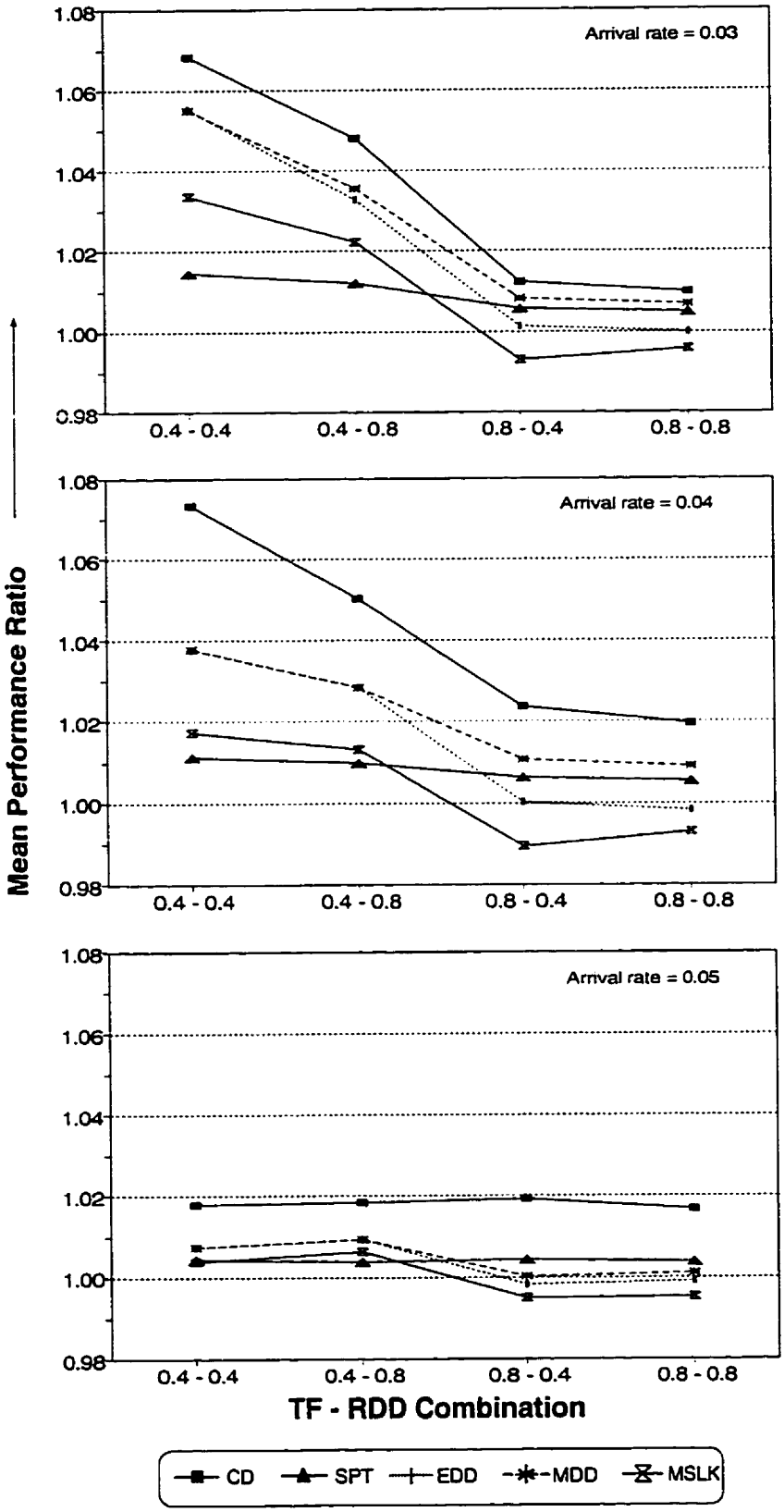


Figure 5.5: Mean tardiness for Type II configuration with FIFO buffers.

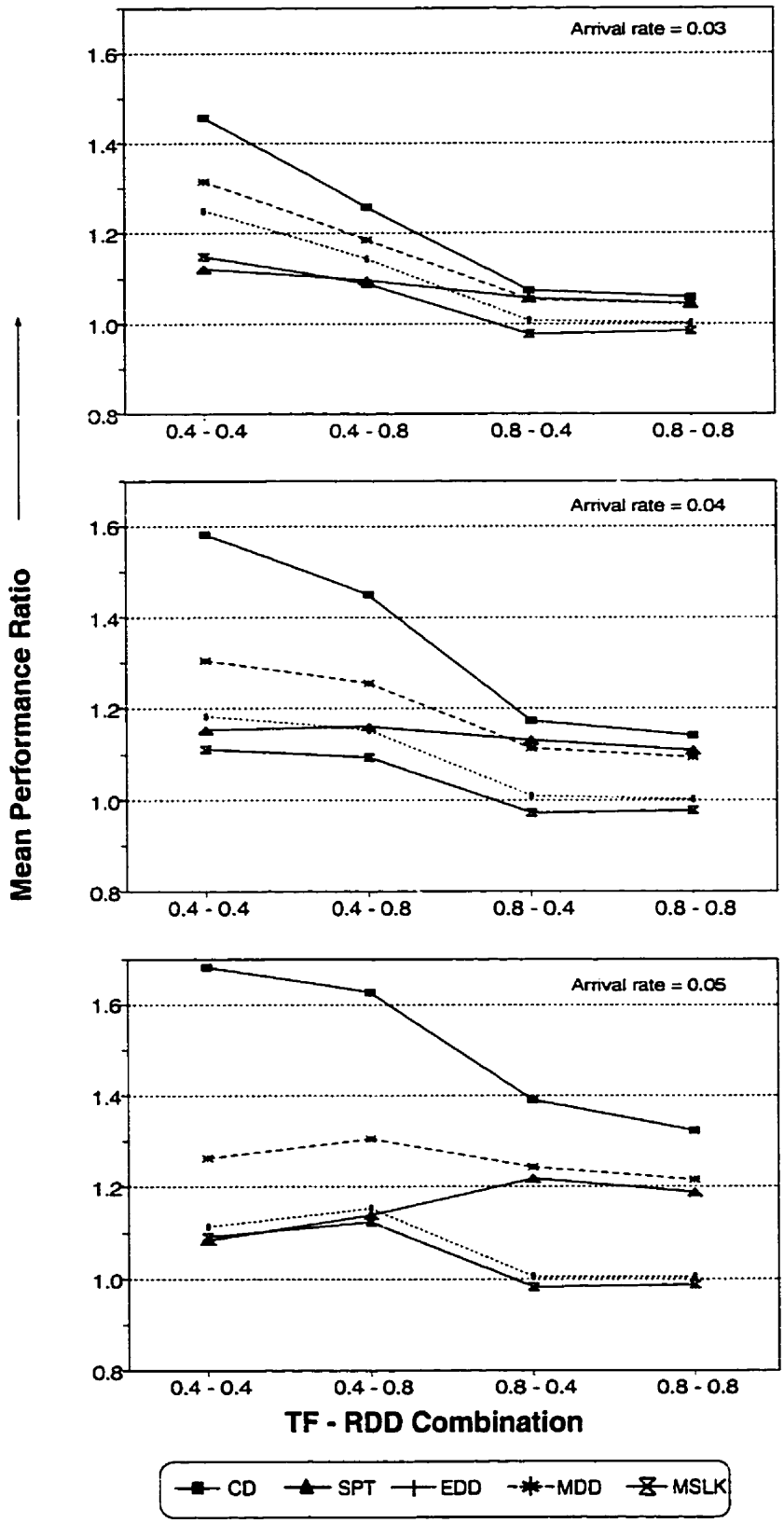


Figure 5.6: Mean tardiness for Type III configuration with FIFO buffers.

Table 5.3: Best solution frequency (in percentage) for mean tardiness.

Configuration	TF	CD	SPT	EDD	MDD	MSLK	FIFO
Type I	0.4	99.17	0.00	0.00	0.00	0.83	0.00
	0.8	97.50	2.50	0.00	0.00	0.00	0.00
Type II	0.4	71.67	9.17	9.17	2.50	1.67	5.83
	0.8	73.33	11.67	2.50	10.83	0.00	1.67
Type III	0.4	97.50	0.83	0.00	1.67	0.00	0.00
	0.8	94.17	5.83	0.00	0.00	0.00	0.00

to give the worst solutions. The results of Table 5.4 show that CD invariably has a mean RDI close to zero. Thus, on this last basis, CD is significantly superior to the other dispatching rules employed in the comparisons.

Table 5.4: Relative Deviation Index for FIFO intermediate buffers.

Configuration	RDI	CD	SPT	EDD	MDD	MSLK	FIFO
Type I	mean	0.000	0.536	0.649	0.350	0.818	0.897
	std. dev.	0.004	0.271	0.245	0.206	0.235	0.126
Type II	mean	0.073	0.557	0.476	0.349	0.694	0.745
	std. dev.*	0.191	0.345	0.327	0.316	0.328	0.297
Type III	mean	0.003	0.443	0.630	0.273	0.795	0.896
	std. dev.	0.019	0.294	0.269	0.166	0.257	0.127

* standard deviation

5.3.3.2 Cells with non-FIFO Intermediate Buffers

Test data is replicated in simulations where the intermediate buffers are not constrained to any selection rule. The corresponding results are presented in Figures 5.7 through 5.9 for the three configurations. They show that, although the margin of superiority of CD is reduced over the other dispatching rules, the differences are still significant in two ways. First, the MDD rule appears to be more dominant across the range of test problems. SPT, on the other hand, is poor for problems having a low TF but stronger than MDD when TF is high. In comparison, CD is consistently superior, by varying margins, to both MDD and SPT. The implication is that, by employing CD, the need to determine which dispatching rule is best for a particular problem is eliminated.

Table 5.5: Best solution frequency (in percentage) for Non-FIFO intermediate buffers.

Configuration	TF	CD	SPT	EDD	MDD	MSLK	FIFO
Type I	0.4	89.17	0.00	0.00	10.83	0.00	0.00
	0.8	81.67	18.33	0.00	0.00	0.00	0.00
Type II	0.4	67.50	0.00	0.00	32.50	0.00	0.00
	0.8	78.33	17.50	0.00	4.17	0.00	0.00
Type III	0.4	82.50	0.00	0.00	17.50	0.00	0.00
	0.8	84.17	15.83	0.00	0.00	0.00	0.00

Secondly, a comparison of the dispatching rules with respect to the number of times each rule finds the best solution (see Table 5.5) reveals that CD outperforms the other rules by a margin largely in excess of 2 to 1. Hence, even in situations where the traditional dispatching rules are at their best, namely non-FIFO intermediate buffers throughout, CD still remains significantly superior. These results are confirmed in Table 5.6 which presents comparisons in terms of the mean and standard deviation of the RDI.

Table 5.6: Relative Deviation Index for non-FIFO intermediate buffers.

Configuration	RDI	CD	SPT	EDD	MDD	MSLK	FIFO
Type I	mean	0.003	0.360	0.658	0.141	0.808	0.909
	std. dev.	0.013	0.328	0.246	0.080	0.223	0.116
Type II	mean	0.005	0.354	0.649	0.092	0.806	0.914
	std. dev.	0.013	0.327	0.261	0.068	0.244	0.110
Type III	mean	0.004	0.352	0.807	0.136	0.797	0.915
	std. dev.	0.013	0.323	0.127	0.090	0.256	0.113

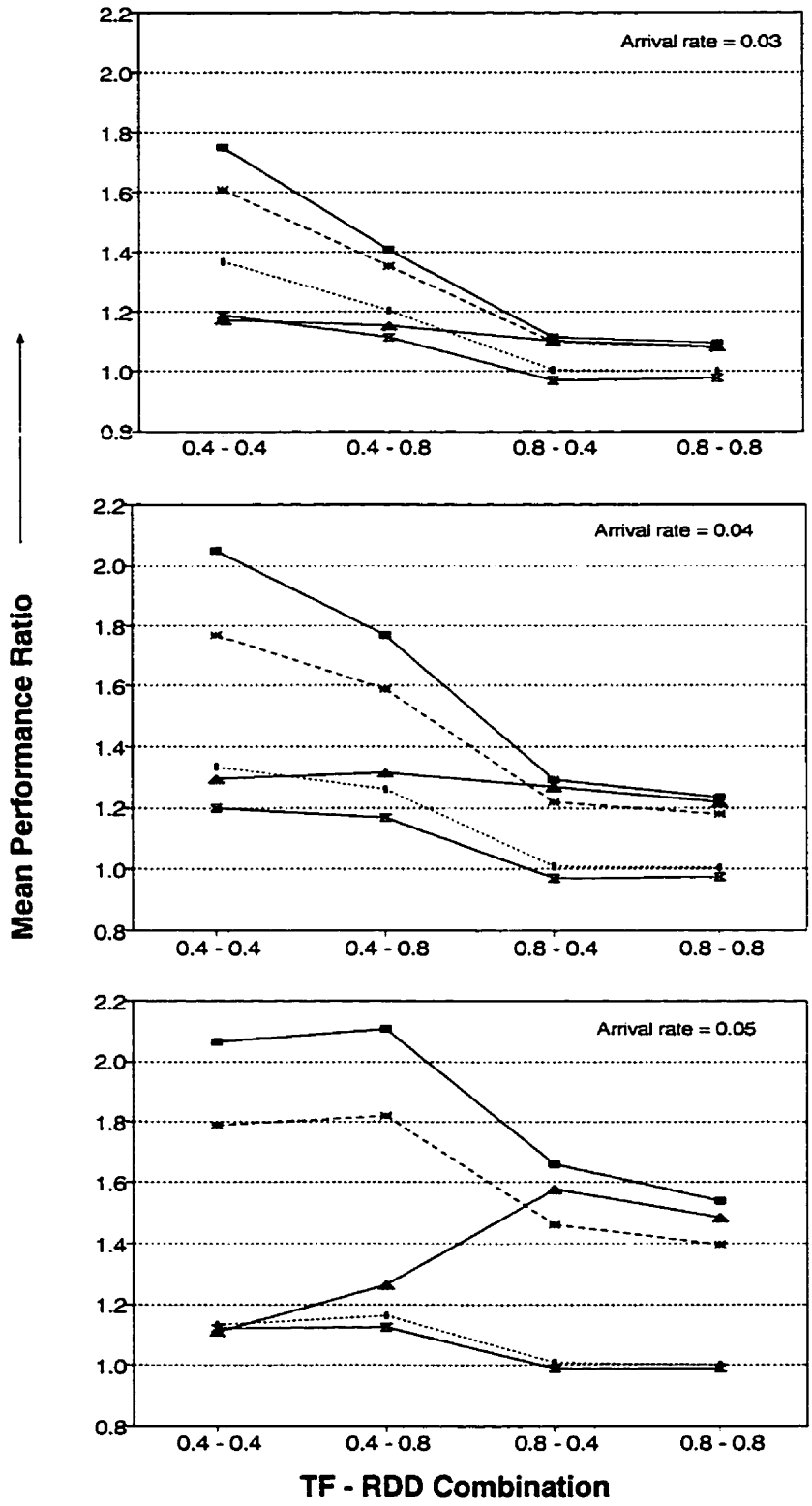


Figure 5.7: Mean tardiness for Type I configuration with non-FIFO buffers.

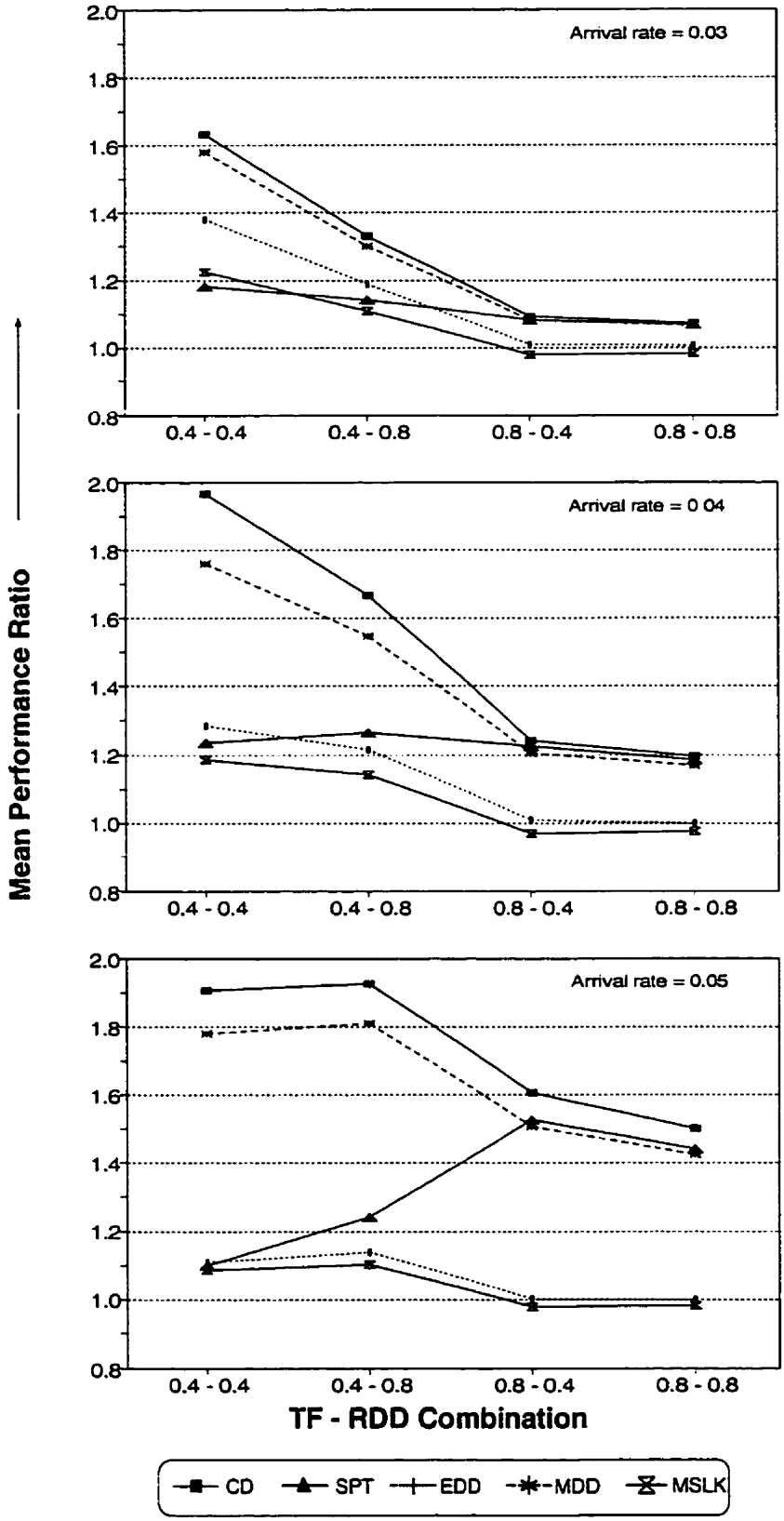


Figure 5.8: Mean tardiness for Type II configuration with non-FIFO buffers.

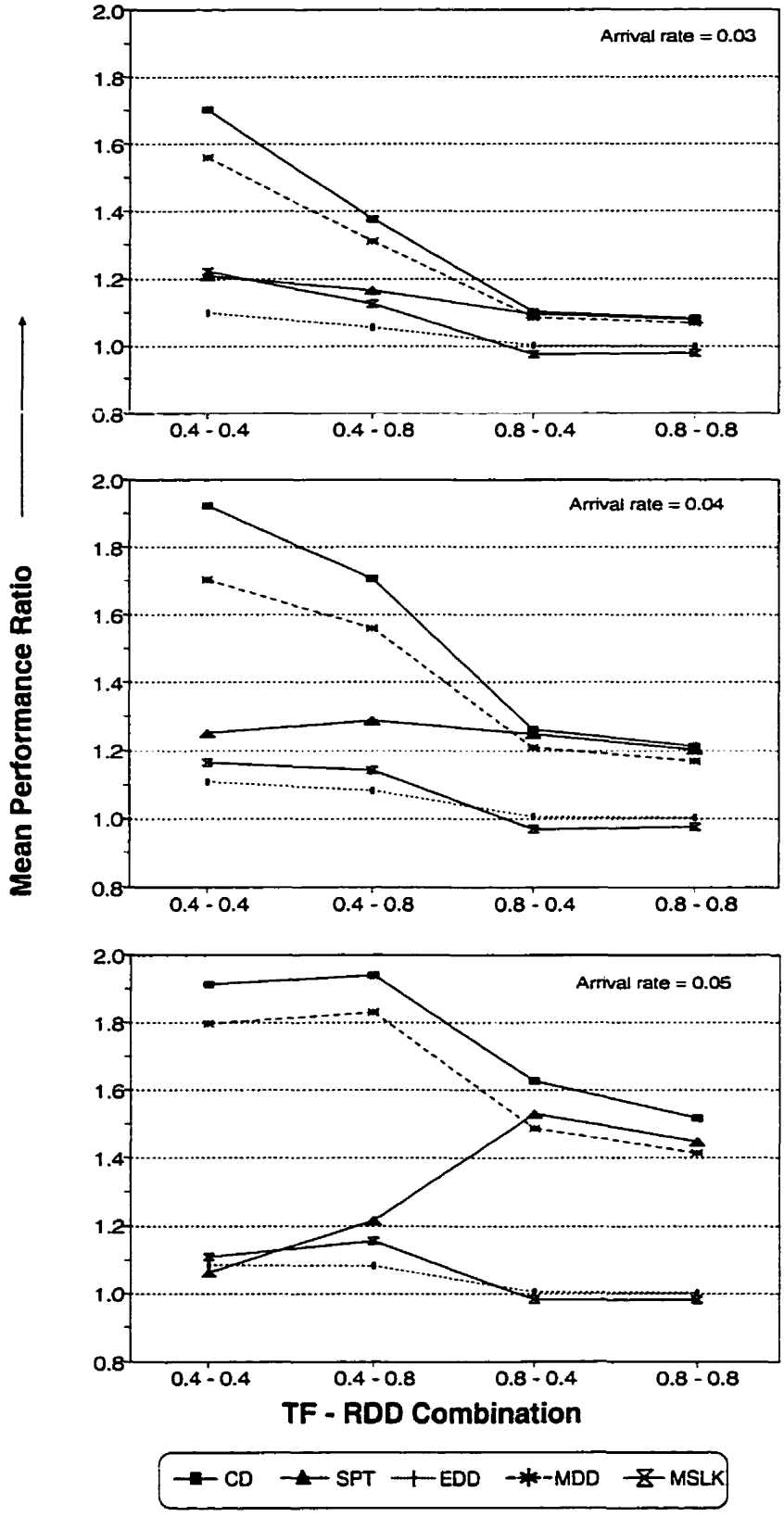


Figure 5.9: Mean tardiness for Type III configuration with non-FIFO buffers.

5.4 Conclusion

The evaluation of CD under conditions of dynamic job arrivals has demonstrated that CD has flexibility with respect to the following factors.

Satisfaction of different performance criteria.
CD performs better than the most obvious 'quick and dirty' rules that can be used for the performance criteria tested.

Adaptability to different configurations.
In all three configurations, CD is generally superior for the different criteria investigated. The overall performance of CD is also observed to be less sensitive to the configuration than the other dispatching rules tested.

Buffer Constraints.
The two extremes of a cell having either all FIFO or all non-FIFO intermediate buffers are investigated. CD excels in both extremes. It is therefore highly probable that CD is also superior in configurations where only a selected number of the intermediate buffers are constrained to FIFO selections. This makes CD attractive in cells whose hardware constraints necessitate the combining of buffers that allow part overtaking with those that do not.

Finally, it may be concluded that the advantages of CD that were identified when it was tested with static problems are also apparent in the dynamic shop scenarios. Thus, CD provides a combination of the algorithmic power of static heuristics with

the high flexibility that is observed in on-line dispatching rules.

Chapter 6

Implementation of CD in an Existing FMC

6.1 Introduction

A scheduling and control system that is based on CD, and its implementation on a real FMC, is presented in this chapter. Chapter 1 described a FMC that is located in the Automation Laboratory at the University of Manitoba. This FMC is the subject of a physical implementation of CD. The main purpose of this actual, real-world application of CD is to demonstrate its feasibility and to identify problems, hardware or otherwise, that can be a major impediment in its use in industrial settings.

6.2 A CD-based Scheduling and Control System

CD's requirement for real-time machine status information is met through the employment of 'agents'. These agents are independent programs that have specific functions. They can communicate with each other as well as with the main control program residing in one of the personal computers. Agents may be classified as 'intelligent' or 'informative'. The agents utilized in the implementation of CD are strictly informative, serving to facilitate a heterarchical relationship between machines. Their primary tasks are to monitor specific activities, and to respond

with the appropriate information when interrogated by the main program that is controlling the CD process.

6.2.1 Informative Agents.

Informative agents are created at the start-up of the FMC control program (i.e. the main program). Each machine is provided with the following three agents.

1. Downloading agent.
2. Monitoring agent.
3. Buffer agent.

The functions of each agent are described next.

- The Downloading Agent.

  This agent receives jobs as they arrive at a machine and it retrieves the corresponding CNC part programs. It supervises the successful downloading of the part program, and informs the Monitoring agent once the machine is ready to start operations. Upon completion of the operations, the Downloading agent consults the Buffer agent of the machine to which the completed part is to be unloaded. Depending on the response, the Downloading agent authorizes the unloading or it waits until a clearance is received from the next machine's Buffer agent.

- The Monitoring Agent.

  The Monitoring agent's main task is to monitor the processing operations on a job. It responds to queries regarding the anticipated time remaining until completion of the current operation on the machine, and it notifies the downloading agent upon successful completion of the current job. The Monitoring agent also compares the actual and expected processing times for a job. If minor deviations are observed, the agent updates its database to reflect the actual processing time for that particular job. Thus, this agent is able to detect trends in a machine's capabilities. Although not considered in this research, such information may be used, say, in a tool wear analysis.

- The Buffer Agent.

  The Buffer agent's function is purely organizational. It tracks the jobs in the buffer, blocks the reception of new jobs if the buffer is filled to capacity, and it collects data for the calculation of instantaneous and overall WIP levels.

6.2.2 The Main Program

The main program runs the CD algorithm and dispatches jobs. It communicates with all the different agents of the machines. The main program is normally in a 'wait' mode, awaiting requests. When a machine is free, its downloading agent sends a message to the main program, requesting to be loaded with a new job. Upon receiving this message, the main program identifies all the jobs that are available for selection and it initiates the CD algorithm. All the Monitoring agents are contacted with requests for the remaining times of the jobs currently being processed. The main program also obtains the number and types of jobs waiting at each buffer from the corresponding Buffer agent. All this information is used subsequently to

determine the machines' ready times, which is needed by the CD algorithm. When a job is selected by the CD, the main program replies to the machine that sent the initial request, giving the identity of the job that will be processed next.

Figure 6.1 illustrates the lines of communication between the different agents and machines, as well as between the programmable logic controller (PLC) and material handling system for the FMC. The operating system that is used is QNX 4.0 [76]. The QNX 4.0 operating system provides an environment that is highly suitable for multi-tasking and networked communications between programs running on different computers and controlling different equipment in the FMC.

6.3 Experimental Trials

The availability of an actual FMC allows the testing of CD under 'real-world' conditions. The robotic handling system found in this FMC is a shared resource so that jobs often have to wait for service. Consequently, a job's waiting time as well as the travel times between the different machines are factors affecting the schedule. The aim of the experiments is to investigate how well CD performs in this kind of situation, considering that the previous computer simulations had assumed these travel times to be negligible. Actual travel time includes travel-empty and travel-loaded times, as well as the pick-up and drop-off times. In the FMC under consideration, the total travel time expended in servicing a part transfer averages between 10 and 12 seconds, depending on the distance between the origin and destination of the travel.

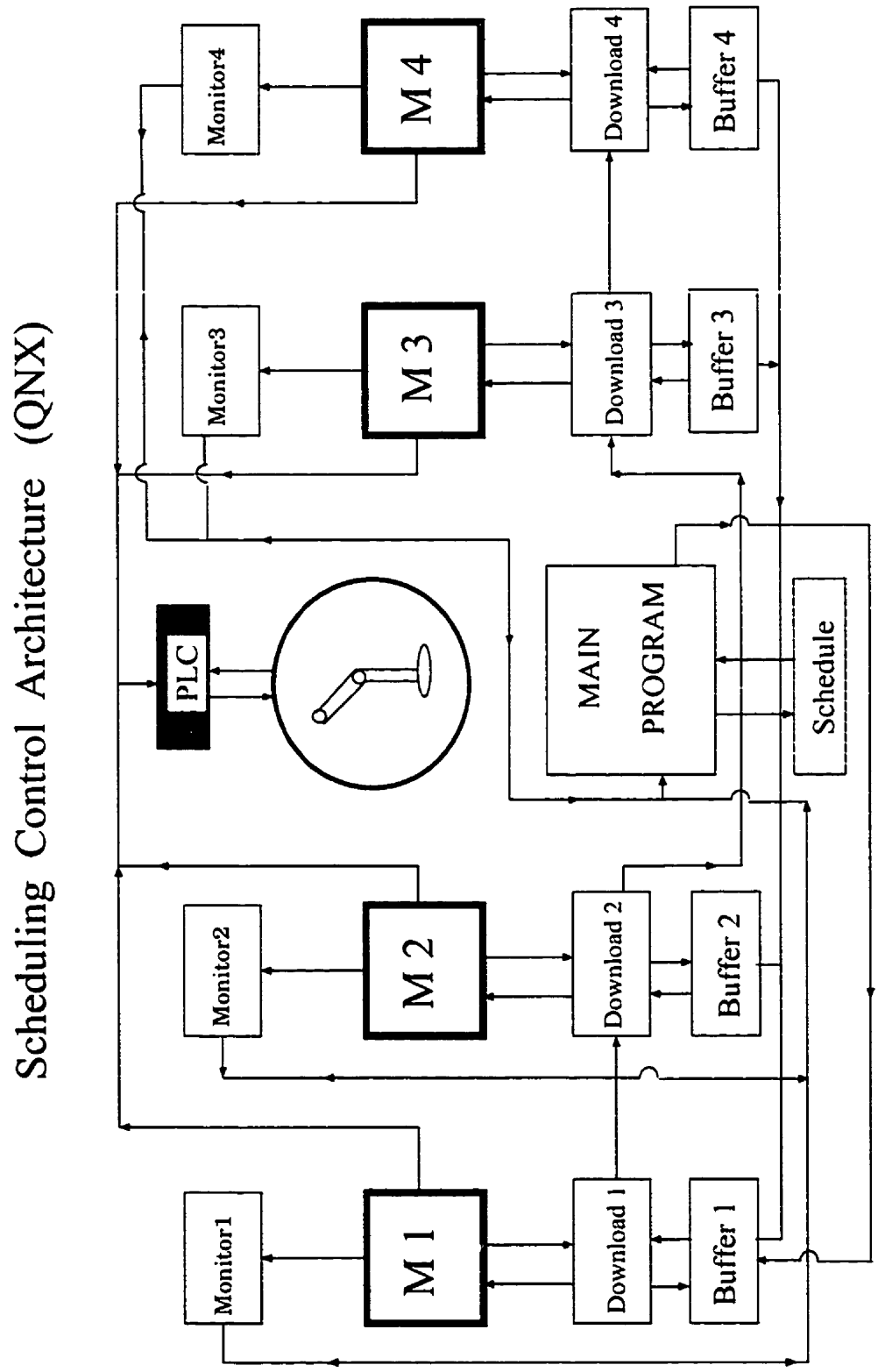


Figure 6.1: Schematic of the control system for the FMC.

6.3.1 Procedure

The FMC contains four machines. The first machine is a TERCO, two and one-half axis mill. The second is also a mill, an EMCO two and one-half axis machine. Part programs are downloaded to these machines from a PC. The downloading time depends on the complexity of the operations on the part, but usually ranges between 25 and 30% of the part's processing time. This relatively high downloading time is due to the old technology used in these aged (1980's) but available machines. The remaining two machines are locally constructed gadgets that are designed to simulate a paint booth and an inspection machine. They go through the motions of the simulated activities, but no processing actually occurs. The FMC has part buffers to load the TERCO and the EMCO mills. The absence of intermediate buffers to hold the WIP means that machine blockage and starvation is a major factor in the performance of this FMC.

Nine parts are manufactured in the experiments. All the parts have the same route, namely Machine 1 (TERCO) to Machine 3 and then to Machine 4. The inclusion of all four machines in the experiments was not possible due to limitations on the available input-output cards used in the PCs. The nine parts are listed in Table 6.1 together with the processing times (in seconds). A given processing time includes only the duration that a machine is physically processing a part. Downloading times are independent of the processing times.

Table 6.1: Processing times (in seconds) for parts manufactured in the FMC.

Part	TERCO (M1)	EMCO (M2)	PAINT (M3)	INSPECT (M4)
A	23	0	39	31
B	54	0	61	90
C	55	0	57	55
D	35	0	80	83
E	55	0	61	85
F	23	0	57	3
G	69	0	3	56
H	23	0	62	3
I	35	0	41	31

6.3.2 Experimental Data

Each test set involves jobs that are selected randomly from the nine parts listed in Table 6.1. Five different sets are created and processed in the FMC. Test set #1 contains one of each part for a total of nine jobs. In each of the remaining four sets, four of the parts shown in Table 6.1, are selected randomly. Three of these parts, which are also picked randomly, have a demand of three units each, and the remaining part has a demand of two units to give a total of eleven jobs in each set. The part types and demanded quantities in each of the last four test sets are given in section 1 of Appendix B.

The performance criterion selected in these experiments is the minimum mean flowtime. The test sets are fed to the FMC in the form of bills-of-materials (B.O.M.)

which list each part demanded and the corresponding due date. In the experiments, a B.O.M. is processed in the cell three times, each time by using one of either the SPT, LWKR or CD rules. The results are compared in Table 6.2. They clearly indicate that CD performs better than the other two dispatching rules. The margins of superiority are similar to those obtained in the simulation experiments discussed earlier in Chapter 3. It is noteworthy that the simulations assume intermediate buffers with unlimited capacity. However, there are no intermediate buffers in the actual FMC. Thus, the results, albeit from a small number of problems, serve to demonstrate CD's ability to accommodate capacity limitations in the intermediate buffers.

Table 6.2: Mean flowtime (in seconds) for the test sets.

Set	SPT	LWKR	CD
1	703	704	658
2	732	772	711
3	819	813	796
4	773	818	742
5	725	718	722

The mean flowtime criterion is selected in the experiments because it better illustrates the essential differences between CD and the traditional dispatching rules. The SPT and LWKR rules, for example, are static because the part selections are always based on part processing times, which are constants. The result is that the order in which the parts flow in the cell is fixed and independent of the time or instantaneous conditions. Furthermore, when considering job batches, the jobs in a

batch normally have identical processing times and due dates. In such cases, rules like SPT and EDD maintain the integrity of the batches. CD, on the other hand, is not averse to breaking up batches into individual units and working effectively with batch sizes of one unit. Table 6.3 illustrates this difference with a comparison of the processing sequences obtained for test data set #4. It is obvious that CD, with its ability to sequentially select individual units from a batched order, utilizes a larger search space that increases the likelihood of finding a better solution.

Table 6.3: Processing sequences for test set #4.

RULE	Processing Sequence
SPT	A - A - A - F - F - F - B - B - B - G - G
LWKR	F - F - F - A - A - A - G - G - B - B - B
CD	A - A - A - F - G - F - G - G - B - B - B

The physical experiments suggest that the CD's theoretical performance holds for real-world situations. In addition, the experience gained from these experiments points to the difficulties that would be faced in building simulation models for these cells. The quantity of messages and signals that are exchanged, together with the possibilities of their nearly simultaneous occurrence, make collecting data for an accurate model very difficult. Finally, one of the main benefits of FMCs is their flexibility that allows production in batch sizes as low as one unit. Traditional dispatching rules tend to promote bigger batching, which defeats one of the main purposes of a FMC (i.e the flexibility to produce efficiently in low unit quantities). CD has the advantages of the dispatching rules but does not compromise the advantages associated with low batch sizes and simultaneous production.

6.4 Conclusions

The implementation of CD in a real FMC is realized by means of a computerized network over which individual programs can communicate with each other and share information in order to collectively meet the scheduling objectives.

A number of experiments designed to test CD's performance in real-world situations were done. In these experiments, CD's calculations were based on real-time job completion and in-process time data. The travel times between the machines, though real, were nonetheless assumed negligible in CD's calculations. Unlike the simulations, however, the effect of these travel times is conveyed to the CD algorithm indirectly through the actual completion times. A more accurate implementation would use an estimate of the anticipated travel time in CD's calculation of the machine ready times (R_k). This modification is likely to boost the CD's performance. The travel time estimates between different destinations can be obtained straightforwardly by maintaining a record of actual travel times during the operations of the FMC.

A major advantage of the modular scheduling control architecture described in this chapter is the ease with which system reconfiguration can be adopted. New machines may be integrated into the scheduler simply by creating new agents and opening the lines of communication between these and the existing agents. Likewise, a machine may be taken out of the system merely by disengaging its agents.

Finally, the ease with which historical data can be collected, processed and then used to modify scheduling procedures gives the CD-based scheduling and control system described in this chapter potential for further flexibility and adaptability in real-world applications.

Chapter 7

Conclusions and Recommendations

One impediment to realizing efficient and automated scheduling in FMCs is the inflexibility of scheduling algorithms [2]. Scheduling systems are usually selected bearing in mind the characteristics of the FMC under consideration. A modification or update of the initial schedules normally requires manual intervention. On-line scheduling utilizing dispatching rules permits greater flexibility from modularization, but there still remains the problem of deciding which rule to use at each machine.

CD is a hybrid algorithm-dispatching rule that is designed to permit a more generic approach to scheduling in FMCs. Its algorithmic nature allows the quick computation of 'good' solutions by reducing the complexity of a problem to a series of single machine problems. When performance criteria are unique so that no method is known to be 'best' for the single machine problem, a neural network can be trained to learn a relationship to determine the job sequences. The dispatching component, on the other hand, imparts flexibility to the algorithm through modularization and localization of the scheduling decisions. Therefore, CD is designed to be generally effective for different scheduling criteria, as well as for different hardware setups or configurations, and for operations under highly dynamic conditions involving the need to constantly update schedules.

The CD approach was introduced in this thesis and its formulation was presented. Its performance was evaluated by means of a series of simulations and tests using randomly generated data. These tests covered three different performance criteria: the minimum mean flowtime, the minimum mean tardiness, and the minimum number of tardy jobs. The performance of CD was also investigated under conditions allowing for part overtaking (i.e. non-FIFO processing in the intermediate buffers). The evaluation of CD was made through comparisons with existing dispatching rules that are identified in the literature to be the most appropriate for each of the scheduling criteria and configurations examined.

Several important conclusions were reached from the particular test evaluations and from observations about CD's general performance. CD is a scheduling system that can be implemented in a non-hierarchically controlled cell. As such, it improves the possibility for greater scheduling flexibility. Although the same can be said for traditional dispatching rules, CD possesses the following advantages over these rules, based on the results from the problems tested.

First, CD generally performs better than traditional dispatching rules in satisfying the given performance criterion. Second, CD eliminates the need to decide which dispatching rule to use and when it is to be used. CD is a self-contained rule that adapts to the existing scheduling criteria and instantaneous conditions. Third, CD is the least sensitive of the scheduling methods and rules examined for the different types of configurations considered. Fourth, CD is able to adjust well to cells that have one or more intermediate buffers restricted to a FIFO policy for processing the jobs waiting in a queue. In comparison, traditional dispatching rules

are sensitive to the ability of intermediate buffers to select from the queue. Moreover, their performance deteriorates rapidly as more of these buffers are restricted to FIFO. Consequently, CD represents a step in the direction of a more 'generic' dispatching rule, one that is less sensitive to hardware configurations and machine setup or layout details. In addition, CD is observed to perform equally well in dynamic as well as static situations.

In conclusion, CD adequately addresses the issue of the inflexibility of algorithms raised in [2] by combining the power of algorithmic solutions with the utility and flexibility of the dispatching rule approach. The result is a scheduling system that meets the challenge of operating in a highly automated environment that is subject to variations and uncertainties in the scheduling demands placed on it. Furthermore, CD is well-suited to application in a non-hierarchical framework, the resulting benefits of which are self-configuration, good adaptability to operational variations and hardware constraints, as well as simplified control software.

7.1 Recommendations

Experiences from the present work allow several recommendations and suggestions to be made concerning possible directions of future research. The recommendations are categorized into those that involve artificial neural networks in scheduling and those that pertain to CD and its concepts.

7.1.1 Artificial Neural Networks

The artificial neural network was used for single machine problems. There are two areas where further research could be useful.

The first recommendation is an application-oriented one. In many industrial and service applications, sequencing decisions are made by human operators on a rule-of-thumb basis. The decision process is vague and not defined clearly. The neural networks proposed in Chapter 4 may be used to learn the decision making function. They can be trained on-line by collecting data and documenting the operator's decisions. Once trained, it would be interesting to see how well a neural network can actually replace the operator.

The second recommendation involves using a neural network as a preprocessor to derive starting solutions for further processing by means of simulated annealing. An initial solution generated by a neural network is subjected to simulated annealing, the purpose of which is to arrive at a sequence of jobs that better satisfies the given performance criterion. The research's objective would be to determine how much faster simulated annealing is able to find a solution within a given percentage of the optimal when its starting sequence (the seed) is supplied by a neural network similar to that described in Chapter 4, rather than by other commonly employed means.

7.1.2 Cooperative Dispatching

The results from CD are encouraging but further work is needed in its application in real-world situations. The following recommendations are suggested with this

objective in mind.

1. The assumption of unlimited buffer sizes was maintained throughout this research. This assumption is often unrealistic because FMCs usually have a finite buffer that has a very limited size. CD needs no modification to enable it to process cases involving finite buffer sizes. In practice, if a buffer is full, the preceding station would be 'machine-blocked' and it would not be calling the dispatching algorithm. There is no reason to expect that CD would be any less effective in cells having buffers of finite size. Therefore, it is suggested to further test and evaluate, by means of simulation experiments, the performance of CD for finite buffer sizes.

2. The assumption that a machine does not wait for any job that is not already in its buffer eliminates many scheduling alternatives. CD can be modified to run without this 'no inserted idle time' constraint. This relaxation is one that should be considered for further research.

3. The number of routing possibilities (configurations) considered is small. Further research with CD is needed in flowshops having a greater number of routing patterns. A possibility is the investigation of CD in a FMS that is composed of individual cells of the three types shown in Figure 3.1.

4. The extension of CD to job shops is worth considering. A drawback is the fact that traditional dispatching rules excel as a shop's routing configurations approach that of a job shop. Nevertheless, CD may yet exhibit strength in this area, particularly if there are restrictions on the sequence of processing job queues at the intermediate buffers.

5. Finally, there is still room to improve the quality of the CD solutions beyond that achieved in this research. Specifically, studies may lead to improved methods for estimating the sequence costs, SC_{ky}, discussed in Chapter 3. Also, the assignment of the weight, W_k, to each machine, k, in the cooperative selection routine can be investigated further with a view to using neural networks to learn what weight to assign to each machine under certain conditions.

Bibliography

[1] Duffie, N. A. and Piper, R. S., 1986. Nonhierarchical control of manufacturing systems. Technical Note, *Journal of Manufacturing Systems*, vol. 5, no. 2, pp. 137-139.

[2] Dudek, R. A., Panwalkar, S. S. and Smith, M. L., 1991. The lessons of flowshop scheduling research. *Operations Research*, vol. 40, no. 1, pp. 7-13.

[3] Stecke, K. E., 1986. A hierarchical approach to solve machine grouping and loading problems of flexible manufacturing systems. *European Journal of Operational Research*, vol. 24, no. 3, pp. 369-378.

[4] Maimon, O. Z., 1986. Real-time operational control of flexible manufacturing systems. *Journal of Manufacturing Systems*, vol. 6, no. 2, pp. 221-231.

[5] Duffie, N. A. and Prabhu, V. V., 1994. Real-time distributed scheduling of heterarchical manufacturing systems. *Journal of Manufacturing Systems*, vol. 13, no. 2, pp. 94-107.

[6] Shaw, M. J., 1988. Dynamic scheduling in cellular manufacturing systems: a framework for networked decision making. *Journal of Manufacturing Systems*, vol. 7, no. 2, pp. 221-231.

[7] Shaw, M. J., 1988. Knowledge-based scheduling in flexible manufacturing systems: an integration of pattern-directed inference and heuristic search. *International Journal of Production Research*, vol. 26, no. 5, pp. 821-844.

[8] Baker, K. R., 1974. Introduction to Sequencing and Scheduling. J. Wiley & Sons, New York.

[9] Conway, R. W., Maxwell, W. L. and Miller, L. W., 1967. Theory of Scheduling. Addison-Wesley, Reading, MA.

[10] Blackstone, J. H., Phillips, D. T. and Hogg, G. L., 1982. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research*, vol. 20, no. 1, pp. 27-45.

[11] Haupt, R., 1989. A survey of priority rule-based scheduling. *OR Spektrum*, vol. 11, no. 1, pp. 3-16.

[12] Sabuncuoglu, I. and Hommertzheim, D. L., 1992. Dynamic dispatching algorithm for scheduling machine and automated guided vehicles in a flexible manufacturing system. *International Journal of Production Research*, vol. 30, no. 5, pp. 1059-1079.

[13] Sabuncuoglu, I. and Hommertzheim, D. L., 1992. Experimental investigation of FMS machine and AGV scheduling rules against the mean flow-time criterion. *International Journal of Production Research*, vol. 30, no. 7, pp. 1617-1635.

[14] Garetti, M., Pozzetti, A. and Bareggi, A., 1990. On-line loading and dispatching in flexible manufacturing systems. *International Journal of Production Research*, vol. 28, no. 7, pp. 1271-1292.

[15] Ro, I. and Kim, J., 1990. Multi-criteria operational control rules in flexible manufacturing systems (FMSs). *International Journal of Production Research*, vol. 28, no. 1, pp. 47-63.

[16] Montazeri, M. and Van Wassenhove, L. N., 1990. Analysis of scheduling rules for an FMS. *International Journal of Production Research*, vol. 28, no. 4, pp. 785-802.

[17] Nakamura, N. and Salvendy G., 1988. An experimental study of human decision-making in computer-based scheduling of flexible manufacturing system. *International Journal of Production Research*, vol. 26, no. 4, pp. 567-583.

[18] Wu, S. D. and Wysk, R. A., 1990. An inference structure for the control and scheduling of manufacturing systems. *Computers & Industrial Engineering*, vol. 18, no. 3, pp. 247-262.

[19] Kusiak, A. and Chen, M., 1988. Expert systems for planning and scheduling manufacturing systems. *European Journal of Operational Research*, vol. 34, no. 2, pp. 113-130.

[20] Kathawala, Y. and Allen, W.R., 1993. Expert systems and job shop scheduling. *International Journal of Operations & Production Management*, vol. 13, no. 2, pp. 23-35.

[21] Nakasuka, S. and Yoshida, T., 1992. Dynamic scheduling system utilizing machine learning as a knowledge acquisition tool. *International Journal of Production Research*, vol. 30, no. 2, pp. 411-431.

[22] Shaw M. J., Park, S. and Raman, N., 1992. Intelligent scheduling with machine learning capabilities: the induction of scheduling knowledge. *IIE Transactions*, vol. 24, no. 2, pp. 156-168.

[23] Chiu, C. K. and Yih, Y., 1995. A learning-based methodology for dynamic scheduling in distributed manufacturing systems. *International Journal of Production Research*, vol. 33, no. 11, pp. 3217-3232.

[24] Cho, H. and Wysk, R.A., 1993. A robust adaptive scheduler for an intelligent workstation controller. *International Journal of Production Research*, vol. 31, no. 4, pp. 771-789.

[25] Wu, S. D. and Wysk, R. A., 1988. Multi-pass expert control system - A control/scheduling architecture for flexible manufacturing cells. *Journal of Manufacturing Systems*, vol. 7, no. 2, pp. 107-120.

[26] Rabelo, L., Yih, Y., Jones, A. and Witzall, G., 1993. Intelligent FMS scheduling using modular neural networks. *Proceedings of the 1993 international conference on neural networks*, vol III. San Francisco, California, pp. 1224-1229.

[27] Fahlman, S. and Lebiere, C., 1990. The cascade-correlation learning architecture. Technical report CMU - CS - 90 - 100.

[28] Ishii, N. and Talavage, J. J., 1991. A transient-based real-time scheduling algorithm in FMS. *International Journal of Production Research*, vol. 29, no. 12, pp. 2501-2520.

[29] Jeong, K.-C. and Kim, Y.-D., 1998. A real-time scheduling mechanism for a flexible manufacturing system: using simulation and dispatching rules. *International Journal of Production Research*, vol. 36, no. 9, pp. 2609-2626.

[30] Min, H.-S., Yih, Y. and Kim, C.-O., 1998. A competitive neural network approach to multi-objective FMS scheduling. *International Journal of Production Research*, vol. 36, no. 7, pp. 1749-1765.

[31] Kohonen, T., 1984. Self-Organization and Associative Memory. Springer-Verlag, Berlin.

[32] Sim, S. K., Yeo, K. T. and Lee, W. H., 1994. An expert neural network system for dynamic job shop scheduling. *International Journal of Production Research*, vol. 32, no. 8, pp. 1759-1773.

[33] Johnson, S. M., 1954. Optimal two- and three- stage production schedules with set-up times included. *Naval Research Logistics Quarterly*, no. 1, pp. 61-68.

[34] Nawaz, M., Enscore, E. E. and Ham, I., 1983. A heuristic algorithm for the *m*-machine, *n*-job flow-shop scheduling problem. *Omega*, vol. 11, no. 1, pp. 91-95.

[35] Park, Y. B., 1981. A simulation study and analysis for evaluation of performance-effectiveness of flowshop sequencing heuristics: a static and a dynamic flowshop model. Master's thesis, Pennsylvania State University.

[36] Taillard, E., 1990. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, vol. 47, no. 1, pp. 65-74.

[37] Koulamas, C., 1996. A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, vol. 105, no. 1, pp. 66-71.

[38] Gupta, J. N. D., 1972. Heuristic algorithms for multistage flowshop scheduling problem. *AIIE Transactions*, vol. 4, no. 1, pp. 11-18.

[39] Miyazaki, S., Nishiyama, N. and Hashimoto, F., 1978. An adjacent pairwise approach to the mean flowtime scheduling problem. *Journal of Operations Research Society of Japan*, vol. 21, pp. 287-299.

[40] Rajendran, C. and Chaudhuri, D., 1992. An efficient heuristic approach to the scheduling of jobs in a flowshop. *European Journal of Operational Research*, vol. 61, no. 3, pp. 318-325.

[41] Ho, J. C., 1995. Flowshop sequencing with mean flowtime objective. *European Journal of Operational Research*, vol. 81, no. 3, pp. 571-578.

[42] Woo, H. and Yim, D., 1998. A heuristic algorithm for mean flowtime objective in flowshop scheduling. *Computers and Operations Research*, vol. 25, no. 3, pp. 175-182.

[43] Garey, M. R., Johnson, D. S. and Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. *Math. Opns.Res.*, vol. 1, no. 2, pp. 117-129.

[44] Sen, T., Dileepan, P. and Gupta, J. N. D., 1989. The two-machine flowshop scheduling problem with total tardiness. *Computers and Operations Research*, vol. 16, no. 4, pp. 333-340.

[45] Kim, Y. D., 1993. A new branch and bound algorithm for minimizing mean tardiness in two-machine flowshops. *Computers and Operations Research*, vol. 20, no. 4, pp. 391-401.

[46] Kim, Y. D., 1993. Heuristics for flowshop scheduling problems minimizing mean tardiness. *Journal of the Operational Research Society*, vol. 44, no. 1, pp. 19-28.

[47] Gelders, L. F. and Sambandam, N., 1978. Four simple heuristics for scheduling a flowshop. *International Journal of Production Research*, vol. 16, no. 3, pp. 221-231.

[48] Osman, I.H. and Potts, C.N., 1989. Simulated annealing for permutation flowshop scheduling. *Omega*, vol. 17, no. 6, pp. 551-557.

[49] Ogbu, F.A. and Smith, D.K., 1990. The application of the simulated annealing algorithm to the solution of the $n/m/C_{max}$ flowshop problem. *Computers and Operations Research*, vol. 17, no. 3, pp. 243-253.

[50] Widmer, M. and Hertz, A., 1989. A new heuristic for the flow shop sequencing problem. *European Journal of Operational Research*, vol. 41, no. 2, pp. 186-193.

[51] Potts C. N. and Van Wassenhove, L. N., 1991. Single machine tardiness sequencing heuristics. *IIE Transactions*, vol. 23, no. 4, pp. 346-354.

[52] Rajendran, C. and Holthaus, O., 1999. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research*, vol. 116, no. 1, pp. 156-170.

[53] Holthaus, O. and Rajendran, C., 1997. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, vol. 48, no. 1, pp. 87-105.

[54] Chan, D. Y. and Bedworth, D. D., 1990. Design of a scheduling system for flexible manufacturing cells. *International Journal of Production Research*, vol. 28, no. 11, pp. 2037-2049.

[55] Du, J. and Leung, J.Y.T. 1990. Minimizing total tardiness on one machine is NP-hard. *Math. Opns. Res*, vol. 15, no. 3, pp. 483-495.

[56] Holsenback, J. E. and Russell, R. M., 1992. A heuristic algorithm for sequencing on one machine to minimize total tardiness. *Journal of the Operational Research Society*, vol. 43, no. 1, pp. 53-62.

[57] Emmons, H., 1969. One machine sequencing to minimize certain functions of job tardiness. *Operations Research*, vol. 17, no. 4, pp. 701-715.

[58] Panwalker, S. S., Smith, M. L. and Koulamas, C. P., 1993. A heuristic for the single machine tardiness problem. *European Journal of Operational Research*, vol. 70, no. 3, pp. 304-310.

[59] Wilkerson, L. J. and Irwin, J. D., 1971. An improved algorithm for scheduling independent tasks. *AIIE Transactions*, vol. 3, pp. 239-245.

[60] Van Wassenhove, L. N. and Gelders, L. F., 1980. Solving a bicriterion scheduling problem. *European Journal of Operational Research*, vol. 4, no. 1, pp. 42-48.

[61] Lin, K. S., 1983. Hybrid algorithm for sequencing with bicriteria. *Journal of Optimization Theory and Applications*, vol. 39, no. 1, pp. 105-124.

[62] Chen, C-L. and Bulfin, R. L., 1994. Scheduling a single machine to minimize two criteria: maximum tardiness and number of tardy jobs. *IIE Transactions*, vol. 26, no. 5, pp. 239-245.

[63] Armentano, V. A. and Roncini, D. P., 1999. Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers and Operations Research*, vol. 26, no. 3, pp. 219-235.

[64] Lee, C.Y. and Choi, J. Y., 1995. A genetic algorithm for job sequencing problems with distinct due-dates and general early-tardy penalty weights. *Computers and Operations Research*, vol. 22, no. 8, pp. 857-869.

[65] Held, M. and Karp, R. M., 1962. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, vol. 10, no. 1, pp. 196-210.

[66] Das, S. R., Gupta, J. N. D. and Khumawala, B. M., 1995. A savings index heuristic algorithm for flowshop scheduling with sequence dependent set-up times. *Journal of the Operational Research Society*, vol. 46, no. 11, pp. 1365-1373.

[67] Moore, J. M., 1968. A n-job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science*, vol. 15, no. 1, pp. 102-109.

[68] Rumelhart, D. E., McClelland, J. L. and the PDP Research Group, 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition. vol. 1. MIT Press/Bradford Books, Cambridge, MA.

[69] Granino, A. K., 1992. Neural Network Experiments on Personal Computers and Workstations. The MIT Press, Cambridge, MA..

[70] Grossberg, S., 1976. Adaptive pattern classification and universal recoding: Part I. parallel development and coding of neural feature detectors. *Biological Cybernetics*, vol. 23, pp. 121-134.

[71] Fry, T. D., Vickens, L., MaCleod, K. and Fernandez, S., 1989. A heuristic solution procedure to minimize $\bar{T}$ on a single machine. *Journal of the Operational Research Society*, vol. 40, no. 3, pp. 293-297.

[72] Koulamas, C., 1994. The total tardiness problem: review and extensions. *Operations Research*, vol. 42, no. 6, pp. 1025-1041.

[73] Russell, R. M. and Holsenback, J. E., 1997. Evaluation of leading heuristics for the single machine tardiness problem. *European Journal of Operational Research*, vol. 96, no. 3, pp. 538-545.

[74] Merten, A. G. and Muller, M. E., 1972. Variance minimization in single machine sequencing problem. *Management Science*, vol. 18, no. 9, pp. 518-528.

[75] Turbo C++, 1990. Borland International Inc. Scotts Valley, CA.

[76] QNX 4 Operating System. QNX Software Systems Ltd. Kanata, Ontario.

Appendix A

Neural Network Data

A.1 MLATENET

Neural Network : MLATENET.

Purpose : To minimize the maximum job lateness.

Configuration : 11-5-1

Training Data : TF $\in$ {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}

RDD $\in$ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}

Training Problems : 1440 (Seed = 2166)

Training Patterns : 4320

Testing Data : TF $\in$ {0.2, 0.4, 0.6, 0.8, 1.0}. RDD $\in$ {0.1, 0.3, 0.5, 0.7, 0.9}

Test Problems : 25 (Seed = 2966)

Test Patterns : 300

Learning Rate : 0.1

Training Epochs : 15,000

TSS : 13.43

eq : 0.635

TD : 0

Weight Matrices

Input - Hidden Layer weights:

```
-3.066990e-001  -9.059555e+000  -1.523147e+000   6.903076e-002
-3.231424e+000   1.025432e-001   1.359400e+001   3.304463e+000
 1.820114e-001  -1.502021e+001   4.853420e-001

 1.022312e-001   7.544243e+000   4.336131e-001   7.073975e-002
 4.668078e+000   5.442970e-001  -1.770292e+001  -1.299319e+000
 9.468389e-002  -7.374052e+000  -2.486156e-001

-2.240847e-001   8.917808e+000   6.972447e+000  -1.976929e-001
-3.857825e-001   1.362066e+000   3.793844e+000   2.133730e+000
-1.216855e-001  -5.046008e-001  -8.792189e-001

 1.833205e-001  -1.413963e+001   5.096458e-001   6.538086e-001
 6.547380e+000   7.121986e-001   3.452943e+000  -2.662191e+000
 5.391429e-001   5.463501e+000  -9.809249e-001

-8.217373e-002   1.547697e+001  -6.119469e-001  -9.438477e-001
-6.336817e+000  -1.214851e+000   4.092513e-001   2.042065e+000
-2.103815e-001   1.618479e+001   9.410969e-001
```

Hidden Layer biases:

```
-4.347391e+000   3.766100e+000  -4.355261e-001   6.533647e+000
-6.855554e+000
```

Hidden - Output Layer weights:

```
-1.106960e+001   2.131350e+000   5.059003e+000  -1.000726e+001
 1.526738e+000
```

Output Layer bias:

```
1.526738e+000
```

A.2 FLONET

Neural Network : FLONET.

Purpose : To minimize the mean weighted flowtime

Configuration : 11-9-1

Training Data : TF $\in$ {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}

RDD $\in$ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}

Training Problems : 1440 (Seed = 2166)

Training Patterns : 4320

Testing Data : TF $\in$ {0.2, 0.4, 0.6, 0.8, 1.0}, RDD $\in$ {0.1, 0.3, 0.5, 0.7, 0.9}

Test Problems : 25 (Seed = 2966)

Test Patterns : 300

Learning Rate : 0.1

Training Epochs : 12.000

TSS : 26.29

eq : 0.859

TD :26

Weight Matrices

Input - Hidden Layer weights:

```
-3.958291e+000   7.972166e-002   4.278783e-001   3.456864e+000
 8.952026e-001  -5.515576e+000   2.316587e-001  -6.146061e-001
 3.455886e+000   9.992399e-001  -8.083598e-002

-1.306666e+001  -1.226466e+000   7.467416e-001  -3.337891e+000
 5.313110e-001  -2.280666e+000  -6.085520e-001   8.554002e-001
-1.160789e+000   3.441786e-001   1.107513e-002

-3.213868e+000  -6.769790e-002   2.033150e-001   8.921369e+000
 2.135010e-001   1.731758e+000   5.910141e-001   2.701317e-001
 1.589236e+000  -4.729265e-001   1.444752e+000

-7.004067e+000   9.713864e-001  -4.368251e-001   2.357204e+000
-1.682129e-001  -3.526396e+000  -8.551941e-001  -4.736033e-001
-4.805099e+000  -1.973866e-001  -2.274358e-002

 1.594817e+000  -4.759546e-001  -1.660939e-002   1.073345e+000
-3.625488e-001   3.906402e+000  -1.036894e+000   1.988397e+000
-2.903833e+000  -7.512914e-002  -3.045168e-001
```

```
-3.145441e+000  -8.646354e-001   5.671426e-001  -8.628493e+000
 1.395264e-001   4.619814e-001  -4.141731e-001  -1.169328e+000
-6.826951e-001   2.874070e-001  -8.104638e-001

-1.112363e+000  -2.872262e+000  -1.019777e+000  -1.497613e+000
 4.467773e-002   2.030772e+000   6.415933e-001   2.140972e+000
-2.698213e+000  -9.851303e-001  -2.488840e+000

-1.194024e+000  -6.495775e-001   7.478958e-001  -1.185082e+000
 3.022461e-001  -1.107189e+000  -2.517221e-001  -7.451938e-002
 1.252505e+000  -1.339409e+000  -1.982472e-001

 6.829798e+000  -1.207234e-001  -2.237389e-001  -5.898614e+000
-6.824951e-001  -6.236802e+000  -1.641974e+000   1.021426e+000
-3.241645e+000  -1.289043e+000  -5.668483e-001
```

Hidden Layer biases:

```
-5.633033e-001   1.203681e+000  -2.543739e+000   4.656348e+000
-4.036040e+000   1.302697e+000  -7.802731e-001   1.906613e-001
 6.289301e+000
```

Hidden - Output Layer weights:

```
 1.074030e+000  -9.927220e+000  -1.632421e+000  -2.487260e+000
-1.058996e+000   6.088645e+000  -2.622602e+000  -8.869069e-001
 1.753356e+000
```

Output Layer bias:

```
8.209095e-001
```

A.3 MTARNET

Neural Network : MTARNET.

Purpose : To minimize the mean tardiness.

Configuration : 11-5-1

Training Data : TF $\in$ {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}

RDD $\in$ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}

Training Problems : 1440 (Seed = 2166)

Training Patterns : 4320

Testing Data : TF $\in$ {0.2, 0.4, 0.6, 0.8, 1.0}. RDD $\in$ {0.1, 0.3, 0.5, 0.7, 0.9}

Test Problems : 25 (Seed = 2966)

Test Patterns : 300

Learning Rate : 0.1

Training Epochs : 50.000

TSS : 54.35

eq : 1.575

TD : 434

Weight Matrices

Input - Hidden Layer weights:

```
 3.422941e+000  -3.290435e+001   1.949022e+001   6.903076e-002
 1.307954e+000  -3.379533e+000   2.319486e+001  -1.067577e+001
-5.188765e-001  -3.643243e+000  -2.834768e+000

 2.245911e+000  -1.239345e+001   1.460653e+001   7.073975e-002
 5.131679e+000  -7.868436e+000   5.915402e+000  -5.663502e+000
 6.157806e-001   4.420018e+000  -1.040275e+001

-6.208619e+000  -4.668241e+000   1.485286e+000  -1.976929e-001
 4.796005e+000  -5.135983e+000  -6.881566e+000   2.405949e+000
 5.511001e+000  -9.238803e+000   3.392075e+000

-2.747124e+000  -4.570393e+000   7.441916e+000   6.538086e-001
 1.988243e+000   4.793335e-001   1.282762e-002  -9.444334e-001
 2.562881e+000   3.001840e+000  -6.650412e+000

-1.345196e-001   1.536452e+001  -8.757015e+000  -9.438477e-001
 2.834704e-001   8.904617e-001  -1.975560e+001   1.299788e+001
-6.304913e-001   9.771422e-001  -4.245096e+000
```

Hidden Layer biases:

```
 1.919859e-001   4.229702e+000   4.746261e+000   1.974510e+000
-2.352674e-001
```

Hidden - Output Layer weights:

```
-1.653063e+000   4.881444e+000   5.629674e-001  -7.084196e+000
 3.285158e+000
```

Output Layer bias:

```
8.815259e-001
```

A.4 MTCLASS

Neural Network : MTCLASS.

Purpose : To categorize a single machine scheduling problem into one of 10 categories.

Configuration : 4-10

Training Data : TF $\in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$

RDD $\in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$

Training Problems : 10,000 (Seed = 2166)

Training Patterns : 10,000

Weight Matrices

0.151939	0.389633	0.627479	0.651928	-0.178366
0.0533543	0.280919	0.935939	0.245317	0.350467
0.453377	0.776992	0.0357607	0.311846	0.590159
0.73834	0.36513	0.439994	0.515783	0.633482
0.106373	0.226577	0.345476	0.897986	-0.103329
0.19769	0.503506	0.828295	-0.30919	0.0390222
0.391582	0.859676	0.246891	0.441876	0.632249
0.58268	-0.0485028	0.0473426	0.142823	0.982013

A.5 Neural Sequencers for Minimizing the Mean Tardiness

Neural Sequencer : MTCAT-1.

Purpose : To minimize the mean tardiness in Category 1 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

 4.677412e-001  -1.137428e+001  -8.409519e+000   6.903076e-002   1.085173e-001
-1.840099e+000  -1.651093e+000  -1.478670e+000   4.215058e-001  -2.775829e+000
 4.516523e+000

 1.436085e-002  -1.552465e+001   3.501302e-001   7.073975e-002  -3.032672e+000
-2.480069e-001   2.438742e+001  -1.829194e+000  -8.815327e-002   1.326606e+001
-4.345551e-001

-6.181824e-001  -1.453786e+001  -3.028842e+000  -1.976929e-001   7.347265e+000
 3.940505e+000  -2.392672e+000   2.026365e+000   2.477845e+000  -2.128601e+001
-1.806759e+000

-2.442908e-001  -1.064676e+001  -1.428081e+000   6.538086e-001  -2.880575e+000
 2.264986e+000   1.114669e+001   5.017054e+000   1.570783e+000   3.381440e+000
-2.583182e-002

-2.504283e-002   1.294635e+001  -1.863931e-001  -9.438477e-001  -4.641526e+000
-1.108663e+000  -6.733944e-001   1.417016e+000  -3.893542e-001   1.099728e+001
 1.416437e+000

-1.431515e-001   9.610055e+000  -6.845273e-001  -7.623291e-001  -2.304063e+000
 1.173246e+000  -4.516498e+000   1.193705e+000   4.433985e-001   3.082200e+001
-4.249310e-001

-5.980269e-001  -1.884919e+000  -3.006337e+000  -6.788330e-001  -4.663624e+000
 2.462074e+000   9.369264e+000   5.615512e+000   8.992622e-001   3.523309e+001
 2.348430e+000

-9.011472e-002   6.173291e+000  -6.212944e-001  -9.266968e-001  -1.385191e+000
-9.779583e-001  -8.671314e-001   5.789083e-001  -4.486903e-001  -1.413402e+000
 4.660978e-001

-1.901171e-001   3.647763e+001  -6.340271e-001   6.187134e-001  -1.970940e+001
-6.100599e-001  -6.557581e-001   2.345549e+000  -6.301678e-001  -2.228032e-001
 1.018702e+000


Hidden Layer biases:

-1.007449e+000  -3.934651e+000   7.297521e+000   -2.894308e+000  -5.160263e+000
-2.589707e+000  -5.248339e+000  -7.530496e-001   -1.943773e+001


Hidden - Output Layer weights:

-3.866877e+000  -1.624008e+000   9.052518e-001  -1.371583e+000   3.071353e+000
1.576229e+001   -1.496721e+001  -1.910170e+000   1.278571e+001


Output Layer bias:

-7.252007e-001
```

Neural Sequencer : MTCAT-2.

Purpose : To minimize the mean tardiness in Category 2 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

-4.435973e-002  -5.784576e-001  -2.839601e-001   6.903076e-002   8.804739e-001
-7.337094e-001   3.008425e-001  -6.317413e-001  -9.312991e-002  -8.062992e-001
-9.202659e-001

-2.178343e-001  -1.078615e+000   2.511777e-001   7.073975e-002   5.003695e-001
 1.253633e-001   1.141004e+000   8.820732e-001   3.446732e-001  -6.369765e-001
 4.784720e-001

-3.073246e-001   7.000551e-001  -9.063693e-002  -1.976929e-001   2.071970e-00
 5.375314e-001   8.723893e-001   7.308593e-001   6.879960e-001   3.160734e-001
 5.015628e-001

 4.030195e-002  -4.750970e-001  -6.756124e-001   6.538086e-001  -1.556723e-001
-5.041401e-002   1.772565e+000  -1.976067e-001  -5.382635e-001   1.153668e+000
 8.298451e-001

 5.283258e-001   4.432214e-001   7.529383e-001  -9.438477e-001  -3.445832e-001
 6.112131e-001  -7.004084e-001   7.256837e-002  -7.649428e-001   9.625859e-001
-3.179390e-001

 1.245621e-001   2.000430e+000   9.023049e-001  -7.623291e-001   1.633510e-001
-1.930649e-001  -2.267479e+000  -2.018901e+000   1.514538e-001   2.900419e-001
 8.584650e-001

-6.420816e-001  -1.843656e+000  -2.336052e+000  -6.788330e-001   7.915624e-002
 4.681961e-001  -8.716483e-002   1.855688e-001   7.079044e-001  -1.177272e+000
-3.239070e-001

-2.568580e-001  -6.491446e-001   5.213666e-001  -9.266968e-001   2.893983e-001
-2.419962e-001   4.890501e-001   5.350408e-001   7.284036e-001  -5.523191e-001
 6.758633e-001

 3.119033e-001   5.030415e-001   9.382060e-001   6.187134e-001  -6.870408e-001
-4.752694e-001  -1.561725e+000   1.266403e-001   6.223078e-001  -8.889450e-001
-1.004289e+000


Hidden Layer biases:

-3.680516e-001  -6.800822e-001   1.007177e-001  -5.653970e-002  -7.016308e-001
 9.212825e-002  -1.952540e-001   8.059094e-001  -4.562845e-001


Hidden - Output Layer weights:

 6.896010e-002  -1.322755e+000  -2.181983e-001  -1.477793e+000   3.935031e-001
 3.322996e+000  -2.450358e+000  -5.770903e-001   1.648837e+000


Output Layer bias:

 2.718521e-001
```

Neural Sequencer : MTCAT-3.

Purpose : To minimize the mean tardiness in Category 3 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

 1.543269e-001  -7.519130e-001  -8.234916e-001   6.903076e-002   6.488763e-001
-9.703837e-001   1.595583e-002  -6.321758e-001  -1.268935e-002  -7.943520e-001
-1.002623e+000

-1.763976e+000  -4.245186e+000   8.267637e-001   7.073975e-002   1.254711e+000
 3.608932e+000   2.203693e+000  -8.665613e-001   7.036184e-003   1.462670e+000
 2.363317e+000

-7.237460e-001   2.103439e-001  -3.511442e-001  -1.976929e-001  -7.450741e-001
 9.131327e-002   3.589182e-001  -2.718264e-002   2.340274e-001   9.640507e-002
 3.178265e-001

 1.448941e+000  -4.842941e-001  -6.458926e-001   6.538086e-001  -6.489238e-001
 3.234526e-002   1.120033e+000  -1.371981e+000  -7.999855e-001   8.357507e-001
 5.856832e-001

-5.728604e-002   2.451279e-001   6.948855e-001  -9.438477e-001  -4.167507e-001
 7.643632e-001  -1.243997e+000   2.368322e-001  -1.234190e+000   1.183098e+000
-4.176374e-001

 2.894936e+000   5.799472e+000   5.793998e-001  -7.623291e-001   3.141943e-001
-1.962564e+000  -6.569283e+000  -8.583745e-001   5.356339e-001   3.690001e+000
 4.693301e-001

 3.444274e+000  -2.214453e+000  -2.013252e+000  -6.788330e-001  -7.619447e-001
 1.881158e+000   8.678598e-002  -1.917318e-001  -4.804392e-001  -1.164534e+000
-1.201869e+000

-1.018096e+000  -1.650474e+000   8.981400e-001  -9.266968e-001  -4.333310e-001
 2.309221e-001   1.851543e-001  -5.259705e-001   4.197438e-001  -2.787046e-001
 8.871562e-001

 8.928878e+000   3.747381e+000  -3.730163e+000   6.187134e-001  -1.895853e+000
-3.530689e+000  -1.977056e+000  -1.611765e+000  -4.746137e+000  -2.441410e+000
 1.875646e+000


Hidden Layer biases:

-4.670905e-001   3.527330e-001  -7.948178e-001  -6.626567e-001  -9.354885e-001
 2.854981e-002  -1.346661e+000   1.988101e-001  -1.624185e+000


Hidden - Output Layer weights:

 6.743297e-001  -2.517326e+000  -3.307031e-001  -9.847968e-001   1.096591e+000
 2.741187e+000  -2.911354e+000  -5.022856e-001   6.054347e+000


Output Layer bias:

 1.890782e-001
```

Neural Sequencer : MTCAT-4.

Purpose : To minimize the mean tardiness in Category 4 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

 4.677412e-001  -1.137428e+001  -8.409519e+000   6.903076e-002   1.085173e-001
-1.940099e+000  -1.651093e+000  -1.478670e+000   4.215058e-001  -2.775829e+000
 4.516523e+000

 1.436085e-002  -1.552465e+001   3.501302e-001   7.073975e-002  -3.032672e+000
-2.480069e-001   2.438742e+001  -1.829194e+000  -8.815327e-002   1.326606e+001
-4.345551e-001

-6.181824e-001  -1.453786e+001  -3.028842e+000  -1.976929e-001   7.347265e+000
 3.940505e+000  -2.392672e+000   2.026365e+000   2.477845e+000  -2.128601e+001
-1.806759e+000

-2.442908e-001  -1.064676e+001  -1.428081e+000   6.538086e-001  -2.880575e+000
 2.264986e+000   1.114669e+001   5.017054e+000   1.570783e+000   3.381440e+000
-2.583182e-002

-2.504283e-002   1.294635e+001  -1.863931e-001  -9.438477e-001  -4.641526e+000
-1.108663e+000  -6.733944e-001   1.417016e+000  -3.893542e-001   1.099728e+001
 1.416437e+000

-1.431515e-001   9.610055e+000  -6.845273e-001  -7.623291e-001  -2.304063e+000
 1.173246e+000  -4.516498e+000   1.193705e+000   4.433985e-001   3.082200e+001
-4.249310e-001

-5.980269e-001  -1.884919e+000  -3.006337e+000  -6.788330e-001  -4.663624e+000
 2.462074e+000   9.369264e+000   5.615512e+000   8.992622e-001   3.523309e+001
 2.348430e+000

-9.011472e-002   6.173291e+000  -6.212944e-001  -9.266968e-001  -1.385191e+000
-9.779583e-001  -8.671314e-001   5.789083e-001  -4.486903e-001  -1.413402e+000
 4.660978e-001

-1.901171e-001   3.647763e+001  -6.340271e-001   6.187134e-001  -1.970940e+001
-6.100599e-001  -6.557581e-001   2.345549e+000  -6.301678e-001  -2.228032e-001
 1.018702e+000


Hidden Layer biases:

-1.007449e+000  -3.934651e+000   7.297521e+000   -2.894308e+000  -5.160263e+000
-2.589707e+000  -5.248339e+000  -7.530496e-001   -1.943773e+001


Hidden - Output Layer weights:

-3.866877e+000  -1.624008e+000   9.052518e-001  -1.371583e+000   3.071353e+000
1.576229e+001   -1.496721e+001  -1.910170e+000   1.278571e+001


Output Layer bias:

-7.252007e-001
```

Neural Sequencer : MTCAT-5.

Purpose : To minimize the mean tardiness in Category 5 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

-3.615809e-001  -1.310852e+000  -4.059140e-001   6.903076e-002   4.160010e-001
-3.588366e-001   1.305026e+000  -2.309809e-001  -7.916819e-001  -1.102245e+000
-1.285914e+000

-5.288227e+000   5.406647e-001  -1.141506e+000   7.073975e-002  -6.577158e-002
-2.197056e+000   6.199761e+000  -1.331330e+000  -2.532996e+000  -4.753310e+000
-9.541688e-001

-7.395977e-001  -6.612605e-002  -7.658251e-001  -1.976929e-001  -7.689396e-001
 1.794291e-001   3.141012e-001   3.205694e-002   1.711026e-002   2.244368e-001
 3.285477e-001

 2.176646e-001  -8.709593e-001  -6.162238e-001   6.538086e-001  -5.929565e-001
 4.143750e-001   2.003421e+000  -1.927774e-001  -1.423689e+000   6.846126e-001
 1.046731e-002

-1.989601e+000   6.692524e-001   5.296493e+000  -9.438477e-001  -1.356290e+000
 2.922846e+000  -5.410731e+000   8.559944e-001  -1.344307e+000   1.091675e+000
-1.611810e+000

 3.665344e+000   5.966870e+000  -1.742053e+000  -7.623291e-001  -1.965971e-001
-2.542975e+000  -2.625420e+000  -3.453919e+000  -2.346335e+000  -4.006563e+000
 3.517250e-001

 5.551236e-001  -6.191207e+000  -2.010188e+000  -6.788330e-001  -1.462144e+000
 2.571602e+000   3.648507e+000   4.931724e+000   1.445998e-001  -2.488296e+000
-3.128461e+000

-8.705059e-001  -1.707396e+000  -1.875214e-001  -9.266968e-001  -5.790188e-001
-4.182576e-001   5.929551e-001   1.703389e-001   2.923881e-002  -9.113238e-001
 2.945211e-001

 1.514818e+000   5.890106e+000   1.777104e+000   6.187134e-001  -2.640531e+000
-3.391398e+000  -2.143056e+000  -5.737065e-001   3.338360e-001  -4.933375e+000
-2.021120e+000


Hidden Layer biases:

-6.999658e-001  -9.677491e-001  -8.186832e-001  -6.066894e-001  -1.875028e+000
-4.822417e-001  -2.046861e+000   5.312232e-002  -2.368863e+000


Hidden - Output Layer weights:

-8.573953e-001  -5.211130e+000  -2.859351e-001  -9.251966e-001   3.346888e+000
 4.901695e+000  -4.488684e+000  -4.851822e-001   4.041551e+000


Output Layer bias:

 1.802884e-001
```

Neural Sequencer : MTCAT-6.

Purpose : To minimize the mean tardiness in Category 6 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

 3.169615e-001  -5.275680e-001  -4.298917e-001   6.903076e-002   8.915110e-001
-8.803287e-001   2.662620e-001  -6.051766e-001  -8.508381e-002  -6.721656e-001
-8.604938e-001

-4.408445e+000  -7.890277e-001   2.822636e+000   7.073975e-002   5.247061e-001
 3.387958e+000   8.403111e-001  -5.466516e-003   1.792683e+000  -7.158598e-001
 1.113712e+000

-4.613308e-001   7.564538e-001   9.647558e-003  -1.976929e-001   2.075261e-001
 5.309547e-001   8.179238e-001   7.088443e-001   6.719116e-001   3.274169e-001
 4.618978e-001

-1.402026e-001  -4.567626e-001  -6.897633e-001   6.538086e-001  -1.830668e-001
 2.266824e-001   1.029329e+000  -7.788115e-001  -7.191585e-001   7.143670e-001
 3.965448e-001

 2.212413e-001   2.776002e-001   7.455743e-001  -9.438477e-001  -3.676277e-001
 4.273477e-001  -5.915722e-001   1.388340e-001  -9.874265e-001   9.138482e-001
-4.295774e-001

 6.505902e-001   2.410887e+000   1.405557e+000  -7.623291e-001   1.992842e-001
-1.999248e+000  -1.144304e+000  -8.694630e-001  -1.278094e-001   1.535338e+000
 1.395318e+000

-1.388960e-001  -6.837067e-001  -1.146175e+000  -6.788330e-001   5.709345e-002
 1.496915e+000  -4.815821e-002   1.619376e-001   7.431228e-001  -1.052811e+000
-2.949699e-001

-1.709127e+000  -4.192469e-001   1.533017e+000  -9.266968e-001   2.857977e-001
 5.813569e-001   4.305713e-001   3.165576e-001   8.921371e-001  -6.867591e-001
 7.567417e-001

 1.990729e-001   1.082375e+000   1.007138e+000   6.187134e-001  -6.529715e-001
-2.080002e+000  -1.441753e+000   6.483853e-001   5.733782e-001   1.086597e-001
-8.974127e-001


Hidden Layer biases:

-2.576805e-001  -4.367163e-001   1.040085e-001  -3.304845e-001  -9.320758e-001
 4.514601e-001  -4.158819e-001   7.699033e-001  -1.155913e-001


Hidden - Output Layer weights:

 6.152979e-001  -4.847762e+000   1.517214e-001  -1.305103e-001   2.455840e-001
 3.562857e+000  -1.500498e+000  -1.150454e+000   2.657597e+000


Output Layer bias:

 1.016191e+000
```

Neural Sequencer : MTCAT-7.

Purpose : To minimize the mean tardiness in Category 7 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

 4.677412e-001  -1.137428e+001  -8.409519e+000   6.903076e-002   1.085173e-001
-1.940099e+000  -1.651093e+000  -1.478670e+000   4.215058e-001  -2.775829e+000
 4.516523e+000

 1.436085e-002  -1.552465e+001   3.501302e-001   7.073975e-002  -3.032672e+000
-2.480069e-001   2.438742e+001  -1.829194e+000  -8.815327e-002   1.326606e+001
-4.345551e-001

-6.181824e-001  -1.453786e+001  -3.028842e+000  -1.976929e-001   7.347265e+000
 3.940505e+000  -2.392672e+000   2.026365e+000   2.477845e+000  -2.128601e+001
-1.806759e+000

-2.442908e-001  -1.064676e+001  -1.428081e+000   6.538086e-001  -2.880575e+000
 2.264986e+000   1.114669e+001   5.017054e+000   1.570783e+000   3.381440e+000
-2.583182e-002

-2.504283e-002   1.294635e+001  -1.863931e-001  -9.438477e-001  -4.641526e+000
-1.108663e+000  -6.733944e-001   1.417016e+000  -3.893542e-001   1.099728e+001
 1.416437e+000

-1.431515e-001   9.610055e+000  -6.845273e-001  -7.623291e-001  -2.304063e+000
 1.173246e+000  -4.516498e+000   1.193705e+000   4.433985e-001   3.082200e+001
-4.249310e-001

-5.980269e-001  -1.884919e+000  -3.006337e+000  -6.788330e-001  -4.663624e+000
 2.462074e+000   9.369264e+000   5.615512e+000   8.992622e-001   3.523309e+001
 2.348430e+000

-9.011472e-002   6.173291e+000  -6.212944e-001  -9.266968e-001  -1.385191e+000
-9.779583e-001  -8.671314e-001   5.789083e-001  -4.486903e-001  -1.413402e+000
 4.660978e-001

-1.901171e-001   3.647763e+001  -6.340271e-001   6.187134e-001  -1.970940e+001
-6.100599e-001  -6.557581e-001   2.345549e+000  -6.301678e-001  -2.228032e-001
 1.018702e+000


Hidden Layer biases:

-1.007449e+000  -3.934651e+000   7.297521e+000  -2.894308e+000  -5.160263e+000
-2.589707e+000  -5.248339e+000  -7.530496e-001  -1.943773e+001


Hidden - Output Layer weights:

-3.866877e+000  -1.624008e+000   9.052518e-001  -1.371583e+000   3.071353e+000
1.576229e+001   -1.496721e+001  -1.910170e+000   1.278571e+001


Output Layer bias:

-7.252007e-001
```

Neural Sequencer : MTCAT-8.

Purpose : To minimize the mean tardiness in Category 8 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

-9.802405e-001  -1.928381e+000  -2.373066e+000   6.903076e-002   2.976796e+000
-2.484656e+000  -2.987442e+000  -1.210596e+000   8.348310e-001   3.232949e-001
-8.876889e-001

-6.828735e+000  -1.358049e+000   1.647144e+000   7.073975e-002  -4.898361e-001
 7.814742e+000   7.793112e-001  -1.147791e+000   5.025221e+000   1.388776e+000
-1.964419e-001

-1.634461e+000   2.348872e-001  -8.343361e-001  -1.976929e-001  -5.431660e-001
 2.490574e-002   6.445853e-002   5.783598e-001   3.015364e-001   2.017782e-002
-8.910426e-002

-3.123145e-001  -5.236168e-001  -7.717883e-001   6.538086e-001  -6.032329e-001
-2.420058e-001   8.475640e-001  -9.693241e-001  -7.899785e-001   6.365144e-001
 2.570358e-001

-5.953310e+000   3.448057e+000   1.624738e+000  -9.438477e-001  -1.896040e+000
 2.592664e+000  -8.693193e-001   1.387335e+000  -6.438985e-001   1.640195e+000
-1.298854e+000

 1.729662e+000   4.081439e+000   1.635250e+000  -7.623291e-001   2.693147e-001
-1.423515e+000  -2.778745e+000   1.612993e-001  -2.380257e+000   4.964108e-001
-7.653250e-001

 7.100394e+000   1.368046e-001  -7.087750e-001  -6.788330e-001  -1.734710e+000
 1.839805e+000   5.197629e-001  -1.451415e+000  -2.560757e-001  -1.461686e+000
-7.934936e-002

-1.628586e+000  -3.585644e-001   1.083534e+000  -9.266968e-001  -6.804573e-002
-4.804588e-002   3.401137e-001  -4.324463e-002   1.354764e+000  -7.247322e-001
 8.215570e-001

 4.592445e+000   2.803810e+000  -2.415856e+000   6.187134e-001  -8.017001e-001
-1.915030e-001  -5.377738e+000   2.593820e+000   2.942798e+000   2.883987e+000
-2.847407e+000


Hidden Layer biases:

 1.860829e+000  -1.391814e+000  -5.929097e-001  -6.169658e-001  -2.414778e+000
-1.632984e-002  -2.319427e+000   5.640954e-001  -5.300326e-001


Hidden - Output Layer weights:

 1.816501e+000  -4.370256e+000   2.831419e-001  -1.575207e-001   2.752506e+000
 1.490720e+000  -2.493593e+000  -1.130141e+000   3.862264e+000


Output Layer bias:

 6.611063e-001
```

Neural Sequencer : MTCAT-9.

Purpose : To minimize the mean tardiness in Category 9 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

-3.765355e-001  -9.278341e-001  -5.303825e-001   6.903076e-002   6.150393e-001
-8.439211e-001   6.211548e-001  -5.088182e-001  -1.777682e-001  -9.558653e-001
-1.024766e+000

-1.737088e-001  -4.656877e-001   1.381375e+000   7.073975e-002  -5.690168e-001
-2.274423e-001   1.698282e+000   1.044472e+000  -2.602542e-001  -9.187117e-001
-3.500135e-001

-3.523689e-001   4.641480e-001  -2.573582e-001  -1.976929e-001  -2.675127e-001
 3.470795e-001   7.059066e-001   5.072762e-001   3.941671e-001   2.131961e-001
 3.225124e-001

-4.998782e-002  -2.027287e-001   3.306529e-002   6.538086e-001  -3.630763e-001
-1.164026e-001   2.375144e+000   1.195874e-001  -8.792536e-001   9.872506e-001
 2.632394e-001

-1.816188e-001   3.510887e-001   5.799499e-001  -9.438477e-001  -3.732517e-001
 4.847368e-001  -1.427347e+000  -5.466904e-001  -7.754613e-001   5.883910e-001
-5.503579e-001

 1.515913e-001   4.008674e+000   2.940300e+000  -7.623291e-001  -5.172030e-001
-7.858254e-001  -2.076621e+000  -7.916463e-001  -3.677728e-001   2.904786e+000
 2.748755e+000

-1.441919e-002  -1.913348e+000  -1.679044e+000  -6.788330e-001   1.640838e-001
 1.697719e-001   2.605315e+000   2.644082e+000   4.145026e-001   1.477061e+000
 7.359818e-001

-1.496150e-001  -8.515130e-001   4.892457e-001  -9.266968e-001  -3.299505e-001
-4.623313e-001   5.107639e-001   4.472873e-001   3.483792e-001  -6.283815e-001
 4.266347e-001

 6.133577e-001   3.741748e+000   3.028686e+000   6.187134e-001  -1.323640e+000
-6.054930e-001  -3.450943e+000  -1.527224e+000   3.406091e-001  -3.749784e+000
-2.553817e+000


Hidden Layer biases:

-5.009275e-001  -1.470994e+000  -3.172564e-001  -3.768092e-001  -8.919895e-001
-8.028476e-001  -4.206330e-001   3.021906e-001  -1.051972e+000


Hidden - Output Layer weights:

-5.576227e-001  -1.744920e+000  -3.966599e-001  -1.365604e+000   7.628268e-001
 4.522039e+000  -3.077816e+000  -4.264953e-001   4.490139e+000


Output Layer bias:

 3.093489e-001
```

Neural Sequencer : MTCAT-10.

Purpose : To minimize the mean tardiness in Category 10 problems.

Configuration : 11-9-1

```
Input - Hidden Layer weights:

 9.138037e-001  -3.734618e-001  -5.651459e-001   6.903076e-002   8.147100e-001
-1.526897e+000   1.768942e-001  -3.543155e-001  -3.374543e-001  -6.866076e-001
-9.861993e-001

-2.307568e+000   5.835626e-002   2.047872e+000   7.073975e-002  -1.497593e-001
 2.692153e+000   1.932154e-001  -5.113264e-001   1.804906e+000  -8.163421e-001
 3.087276e-001

-4.229246e-001   7.710511e-001   3.117959e-002  -1.976929e-001   1.218475e-001
 5.651955e-001   8.134502e-001   6.871363e-001   7.051865e-001   3.164861e-001
 4.569260e-001

 1.793196e-001  -2.212966e-001  -8.016776e-001   6.538086e-001  -3.398076e-001
-2.837144e-001   8.280534e-001  -7.871493e-001  -6.145079e-001   8.727824e-001
 3.352573e-001

-2.000272e-001  -1.812539e-001   6.851012e-001  -9.438477e-001  -3.856572e-001
 1.037912e+000  -3.248384e-001   2.782169e-002  -1.264715e+000   7.548330e-001
-3.428977e-001

 1.906902e+000   2.169696e+000  -3.429744e-001  -7.623291e-001   6.136283e-002
-2.161566e+000  -8.709780e-001  -3.674838e-001   1.001698e+000   2.859245e-001
 9.117205e-001

-2.955782e-001  -5.276509e-001  -1.087544e+000  -6.788330e-001   2.648341e-001
 1.173524e+000  -3.773624e-001  -1.635716e-001   7.783071e-001  -5.621962e-001
-2.059868e-001

-1.074425e+000  -5.237821e-001   8.682619e-001  -9.266968e-001   1.519531e-001
 3.957972e-001   3.978442e-001   1.961179e-001   1.142288e+000  -6.876676e-001
 6.575273e-001

 1.170494e+000   1.115086e+000   8.728573e-001   6.187134e-001  -4.980590e-001
-2.006431e+000  -1.307211e+000   1.004562e+000   8.926166e-001  -1.405179e-001
-5.168353e-001


Hidden Layer biases:

-3.012568e-001  -1.051737e+000   7.210384e-002  -3.535406e-001  -9.043950e-001
-2.242817e-001  -3.198827e-001   7.840943e-001  -2.263915e-001


Hidden - Output Layer weights:

 1.185505e+000  -3.516281e+000  -2.995844e-001   1.460121e-002  -6.149121e-001
 3.005931e+000  -1.281559e+000  -1.150462e+000   2.374844e+000


Output Layer bias:

 4.345924e-001
```

Appendix B

Test data for FMC experimental trials

Problem data for the experimental tests of CD on the FMC at the University of Manitoba:

Table B.1: Data set 2.

Part	Quantity
A	3
I	3
H	3
B	2

Table B.2: Data set 3.

Part	Quantity
I	3
F	3
E	3
B	2

Table B.3: Data set 4.

Part	Quantity
A	3
F	3
C	3
G	2

Table B.4: Data set 5.

Part	Quantity
A	3
I	3
C	2
D	3