HIERARCHICAL CONTROL OF PRODUCTION IN FLEXIBLE MANUFACTURING SYSTEMS

by

JOSEPH GITHU KIMEMIA

B.S. University of Sheffield
(1975)

M.S. Massachusetts Institute of Technology
(1979)

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

April 1982

Signature of Author _____

Department of Electrical Engineering and Computer Science
April 30, 1982

Certified by _____

Dimitri Bertsekas
Thesis Supervisor

_____

Stanley B. Gershwin
Thesis Supervisor

Accepted by _____

Chairman, Departmental Committee on Graduate Students

HIERARCHICAL CONTROL OF PRODUCTION IN FLEXIBLE MANUFACTURING SYSTEMS

by

JOSEPH GITHU KIMEMIA

Submitted to the Department of Electrical Engineering
and Computer Science in partial fulfillment of the
requirements for the Doctor of Philosophy Degree
in Electrical Engineering

ABSTRACT

The problem of controlling production in an automated flexible manufacturing system (FMS) is described. The system consists of machines which can perform a variety of operations on a family of parts. The system also has a transport mechanism which delivers parts to and from machines, and internal and external storage buffers. Computers supervise the operations of all the elements of the FMS. The chief difficulty faced by the control system is to meet production requirements for the parts while the machines fail and become repaired at random times.

A multi-level hierarchical control algorithm is proposed. A flow control level continuously regulates the production rate for the part family. The paths that the parts follow within the system are chosen by the routing level of the algorithm. A sequence controller dispatches parts into the system so that the production and path flow rates are maintained.

The flow control level is formulated as an optimal control problem for a system subject to sudden change of structure. It is shown that optimal policies are feedback laws which are piece-wise constant. Sub-optimal control schemes which approximate the optimal policy are developed. Lower levels of the hierarchy guarantee that the production rates chosen by the flow control level are achieved.

A simulation model based on a planned electronic components assembly system is used to test the effectiveness of the hierarchical controller. Simulation results show that the multi-level algorithm is effective at meeting the demand on the FMS with low inprocess inventory provided that the production requirements are within the effective capacity of the system. A simple computational procedure for determining the effective capacity is developed.

Thesis Superviors: Dr. Dimitri Bertsekas
Title: Professor of Electrical Engineering

Dr. Stanley B. Gershwin
Title: Lecturer & Principal Research Scientist

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisors, Professor Dimitri Bertsekas and Dr. Stanley B. Gershwin for their encouragement, support and many helpful suggestions while I was doing the research. I would also like to express my graditude to the members of the thesis committee, Professors Michael Athans and Nathan Cook for their interest in my work and helpful comments.

I much appreciate the time taken by Michael Kutcher to explain some of the shop-floor problems in manufacturing.

Thanks also go to my office mates, Leon Ekchian, Susan Hall and Paul Wiley, for many useful discussions. The same goes for Ellen Hahne and John Tsitsiklis. The staff and students at L.I.D.S. have given me a lot of support and help. For that, I am grateful.

Thanks to Lisa Babine for typing the thesis and to Arthur Giordani for preparing the figures.

This work was done with support from the NSF under grants APR 76-12036. DAR 78-17826 and ECS 79-20834, Fellowship and Teaching Assistantship support from the Department of Electrical Engineering and Computer Science.

DEDICATION

This thesis is dedicated to my parents

William and Mary Kimemia

and sisters

Caroline, Evelyn, Jane, Angela and Aida

TABLE OF CONTENTS

		PAGE
ABSTRACT		ii
ACKNOWLEDGEMENTS		iii
DEDICATION		iv
TABLE OF CONTENTS		v
CHAPTER I	INTRODUCTION	1
	1.1 Examples of Flexible Manufacturing Systems	3
	1.2 Problem Definition and Outline of Thesis	10
	1.3 Literature Survey and Related Problems	14
CHAPTER II	PRODUCTION CONTROL IN AN FMS	20
	2.1 Introduction	20
	2.2 Flow Control	25
	2.3 The Routing Algorithm	28
	2.4 The Sequence Controller	29
	2.5 Practical Considerations in the Implementation of the Hierarchical Controller	31
	2.6 Summary	34
CHAPTER III	FLOW CONTROL POLICIES FOR THE AUTOMATED WORKCENTER	35
	3.1 Introduction	35
	3.2 The Model for the Flow Controller	38
	3.3 Characterization of the Optimal Policy for the Finite Horizon Problem	43
	3.4 The Continuous Time Discounted Cost Control Problem	50
	3.5 The Minimum Average Cost Problem	53
	3.6 Summary	58

CHAPTER IV	SUB-OPTIMAL CONTROL OF THE FMS	59
	4.1 Introduction	59
	4.2 Controllers that are Open Loop with Respect to the Buffer State	60
	4.3 An Estimate Based (EB) Control Scheme	65
	4.3.1 The Approach	65
	4.3.2 Calculation of the Estimates $\Psi(x,i,t)$	68
	4.3.3 The Infinite Horizon Case	72
	4.3.4 Implementation of the Estimate Based Controller	76
	4.4 The Certainty Equivalence (CE) Controller	77
	4.5 Summary	82
CHAPTER V	PRODUCTION PLANNING FOR AN AUTOMATED MANUFACTURING SYSTEM	84
	5.1 Introduction	84
	5.2 The Equivalent Production Capacity Set	85
	5.2.1 The Infinite Horizon Case	85
	5.2.2 The Finite Horizon Case	88
	5.3 A Production Planning Scenario Using the Equivalent Production Capacity Set	91
	5.4 Summary	93
CHAPTER VI	SIMULATION RESULTS	95
	6.1 Introduction	95
	6.2 The Manufacturing Systems	96
	6.3 The Simulation Model	99
	6.4 Results for a Two-Part Three-Stage Example	101

6.4.1 Introduction 101

6.4.2 Results for the Estimate Based Controller 106

6.4.3 Results for the Certainty Equivalence Controller 112

6.4.4 Characteristics of Estimate Based (EB) and Certainty Equivalence (CE) Controller) 117

6.4.5 Conclusion 122

6.5 Four-Stage Four-Part Example 123

6.5.1 Production Planning 123

6.5.2 Simulation Results for Four-Stage Four Part Example 128

6.5.3 Conclusion 141

6.6 Summary 143

CHAPTER VII CONCLUSION AND GENERALIZATION 145

7.1 Introduction 145

7.2 Summary of the Thesis 145

7.3 Workcenter Modelling and Computation 148

7.3.1 The Failure Model 148

7.3.2 Inspection and Part Rejection 153

7.3.3 Reduction of Computational Effort at the Flow Control Level 156

7.3.4 Improvement of the Sequence Control Level 158

7.4 The Overall Production Planning Problem 159

REFERENCES 164

APPENDIX A PROPERTIES OF OPTIMAL POLICIES 169

APPENDIX B A HEURISTIC OPERATING RULE 179

APPENDIX C THE SIMULATION CODE 184

CHAPTER I

1.0 INTRODUCTION

It is estimated that between 50% and 75% of U. S. annual expenditure on manufactured parts is for items with an annual demand of less than 100,000 units [Cook, 1975]. There has been concern recently about the low productivity of the medium production plants that produce many of these parts.

Transfer lines have long been used for the manufacture of large volumes of single items. Changing over a transfer line to the production of a different item may not be technically feasible or may be costly in terms of lost production. Job shops which produce many different parts in small quantities have in the last twenty years turned to numerically controlled (NC) machine tools especially in the aerospace industry. Numerically controlled machines are versatile and can perform a wide range of operations on a work-piece under computer control.

Flexible manufacturing systems (FMSs) have recently been introduced in an effort to increase the productivity of that sector of industry that produces a medium volume of a set of related parts. Annual production ranges from 200 - 20,000 parts per year [Hughes et al., 1978] which is not high enough to warrant the use of a set of dedicated transfer lines. On the other hand, job shops suffer from a low utilization of expensive machines and a high in-process inventory [Cook, 1975]. An FMS increases productivity by simultaneously reducing the inventory and increasing the utilization of the machining centers.

An FMS consisting of (a) a number of workstations at which operations are carried out on a variety of workpieces, (b) a materials handling system which transports parts to and from the workstations and (c) one or more computers which control the workstations and transportation system. Depending upon the production level and the variety of parts to be produced, the stations may be made up of standard NC machining centers or more process specific units [Hughes et al., 1978]. The stations are flexible in the sense that they can perform different operations on a variety of parts in a random sequence. An integrated system consisting of workstations, transportation system and control computers is also referred to as a workcenter [CAM-I, 1982].

FMSs or flexible workcenters are typically controlled by a hierarchy of computers [Lerner, 1981]. A master control unit supervises production and monitors the state of the transportation system and workstations. A direct numerical control (DNC) computer supervises the operations of the workstations. It is responsible for maintaining part processing programs and downloading them to local computerized numerical control (CNC) computers at each workstation.

An FMS produces a family of parts which are related by similar operational requirements. The parts are manufactured simultaneously, thus avoiding the need to produce and store individual part types in batches. The parts are fed into the system at a loading station and undergo a specified sequence of operations at the workstations before leaving at an unloading station. The flexibility of the system allows some of the parts the choice of more than one station for each operation. This allows production to continue even when a work-station is out of service due to failure or maintenance.

In this thesis, we develop production control algorithms which enable the workcenter controller to satisfy specified requirements for a part family while keeping a low level of in-process inventory. To achieve this objective, production control policies are developed which take into account workstation failures and which exploit the flexibility of workstations to route parts around failed machines.

1.1 Examples of Flexible Manufacturing Systems

The Caterpillar Tractor Company and The Allis Chalmers Corp., operate flexible manuacturing systems for the production of housings for tractor power train components. In both cases, the FMS produces a family of parts simultaneously. The raw material are castings, and the finished products are housings ready for assembly. A similar system manufactures crank casings for light aircraft engines at the Avco-Lycoming plant in Williamsport, PA. The differences between the three plants are in the machines, tooling, control programs and the fixtures.

The Allis Chalmers system illustrated in Figure 1.1 installed in Milwaukee, Wisconsin is currently capable of randomly processing eight different part types with a total annual production of 20,000 parts [Hughes et al., 1978]. There are ten numerically controlled workstations and parts are delivered to and from the stations by carts towed by a chain recessed in the floor. Control is exercised by two mini-computers. The direct numerical control (DNC) computer contains the manufacturing programs and monitors workstation performance. When a part enters a workstation, the correct machining program is automatically loaded into the station controller by the DNC computer. The second computer is the system controller. It runs the entire system (including the DNC computer).

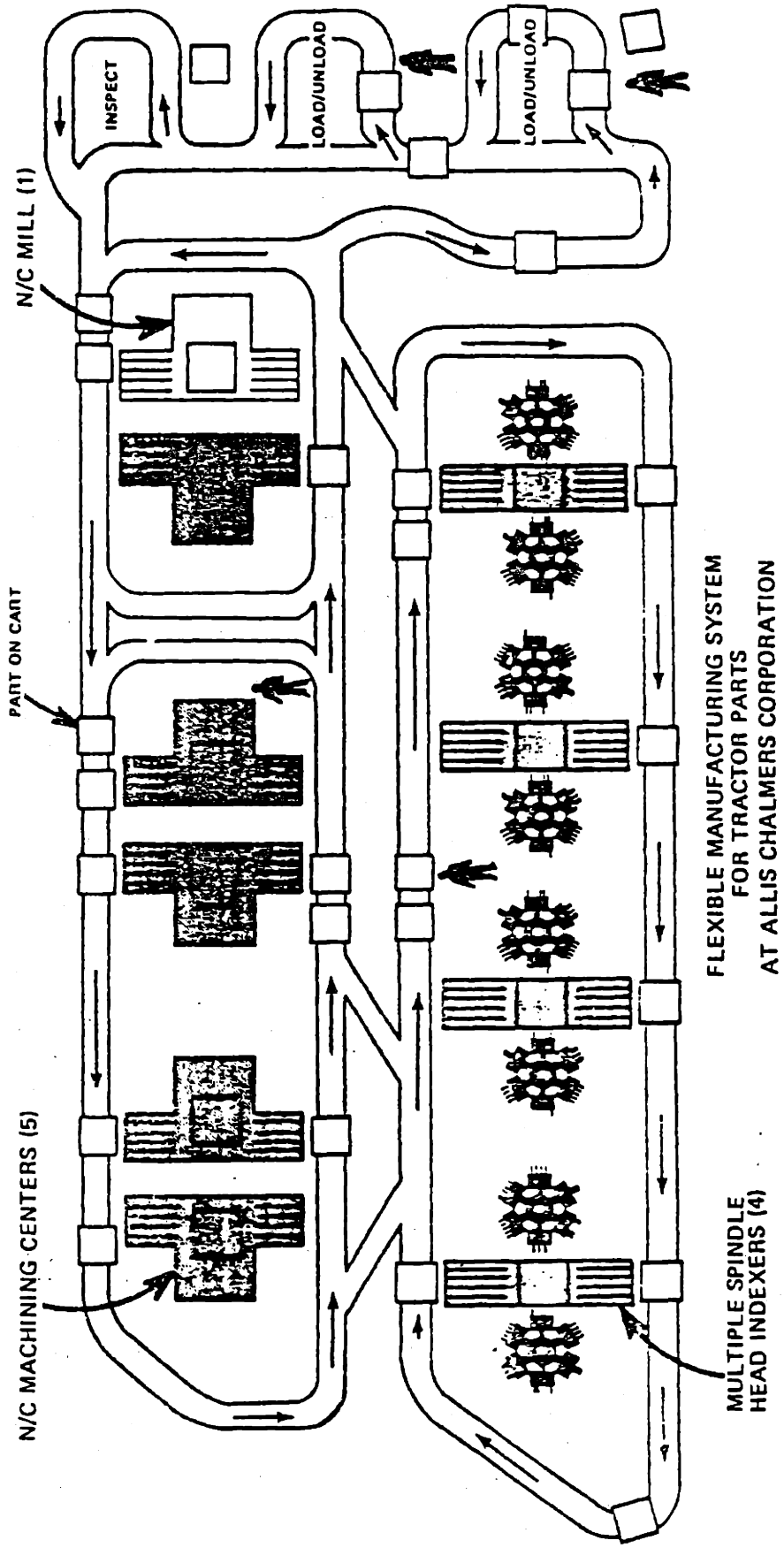

Fig. 1.1. An Example of a Flexible Manufacturing System

It controls all material movement within the FMS, monitors the location and content of each cart on the transportation system and issues load and unload commands to the loading stations.

The flexibility of the system has proved to be very useful. The demand pattern for tractors was different from that originally anticipated. As a result, the part mix produced by the system is different from the one originally planned.

The Caterpillar system shown in Figure 1.2 at Peoria, Illinois makes two families of transmission housings, each consisting of a case and a cover [Stecke, 1977]. The system has ten NC machining centers arranged linearly along a 250 foot track. Routing flexibility is provided by duplicating tools at different stations. The system is controlled by a single Digital Equipment Corporation PDP 10/20 computer. The load-unload area also serves as a centralized buffer for any pieces waiting service at a workstation. The FMS advantage over a transfer line was proved when a major design change was made on one of the parts. The FMS accomplished this by a change in the software and a slight modification of a fixture.

Work piece handling by means of robots is becoming more prominent [Dallas, 1979 ; Winship, 1979]. Robots can be regarded as highly flexible material handling devices. Development of sensors which enable the position and orientation of work pieces to be determined and which allow necessary adjustments to be made to the arms trajectories will no doubt increase the flexibility and areas of application of robots.

The use of robots for part handling is illustrated in the system of Figure 1.3 which makes pinion gears for Massey Ferguson, Inc. tractors in

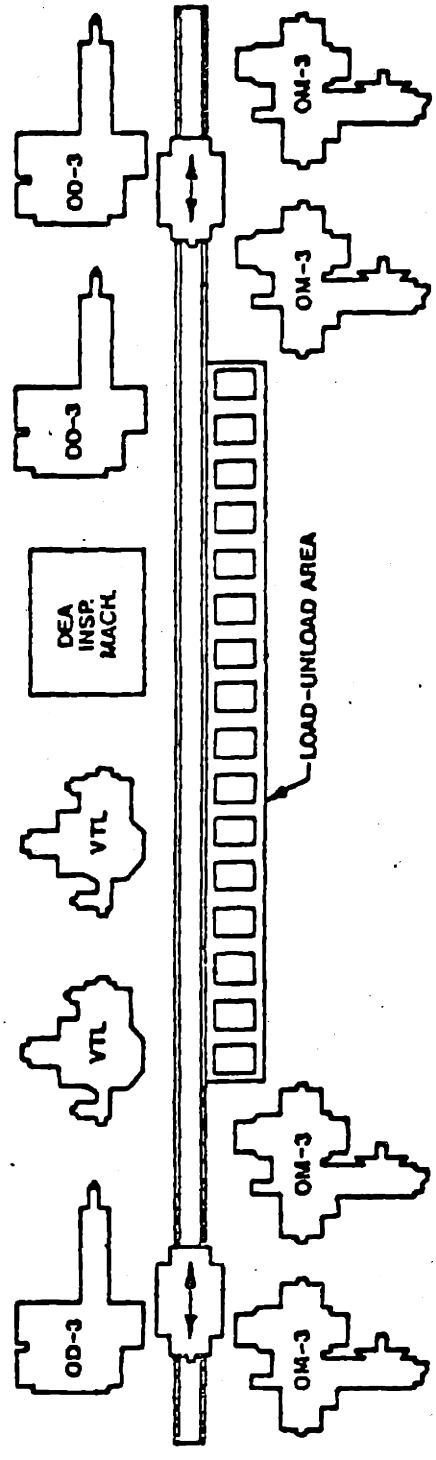

Figure 1.2

The Caterpillar DNC Line

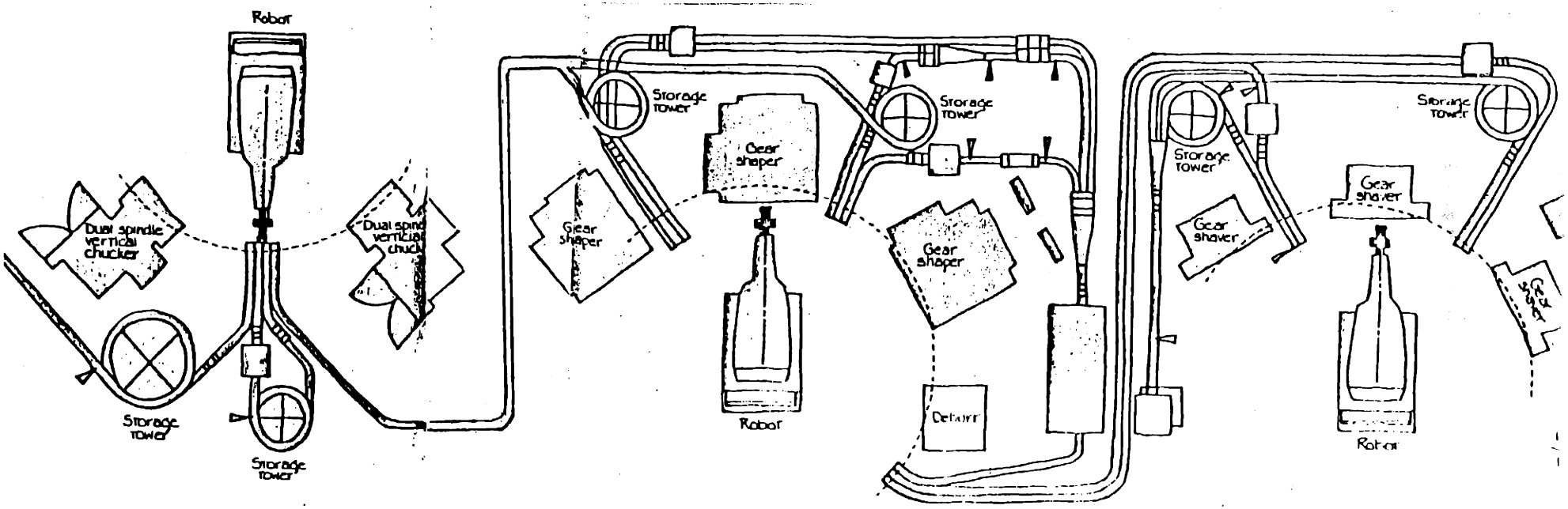

Figure 1.3 The Massey Ferguson Gear Pinion Flexible Transfer Line

Detroit, Michigan. It is referred to as a flexible transfer line since parts visit the stations in a fixed order. The system consists of three workcenters each served by a robot. Four sizes of gear pinions are produced in batches. Flexibility of the system simplifies changeover between batches. The failure of a single machining center does not halt production as in traditional transfer lines. The robots are programmed to avoid failed stations and to load only working stations. The use of robots to load and unload the workstations enabled the designers to specify standard machines, saving time and cost because special handling devices were not required [Dallas, 1979].

Another example of a flexible transfer line using robots for part handling is provided by the Xerox plant in Webster, New York illustrated in Figure 1.4. Fuser rolls for several models of duplicating machines are made. The annual production volume is about 130,000 units [Schaffer, 1978; Gershwin et al, 1980]. All machining and robot handling programs are stored in the central control computer allowing easy change over from one part to another.

Flexible assembly systems are another area of development. In the automobile industry, robots have been used for some time in welding car bodies and spray painting. The General Motors Corporation has been experimenting with a programmable universal machine for assembly (PUMA) [Beecher and Dewar, 1979]. As conceived, the system would be capable for handling and assembling a family of parts. The Japanese motor firm of Nissan has perhaps the most advanced assembly system [Economist, 1980]. Forty-five variations of two models are produced on the same production line with a third model soon to be added. There are 74 automatic and 12 manual stations spread out on five lines with 96% on the welding being done at automated stations.

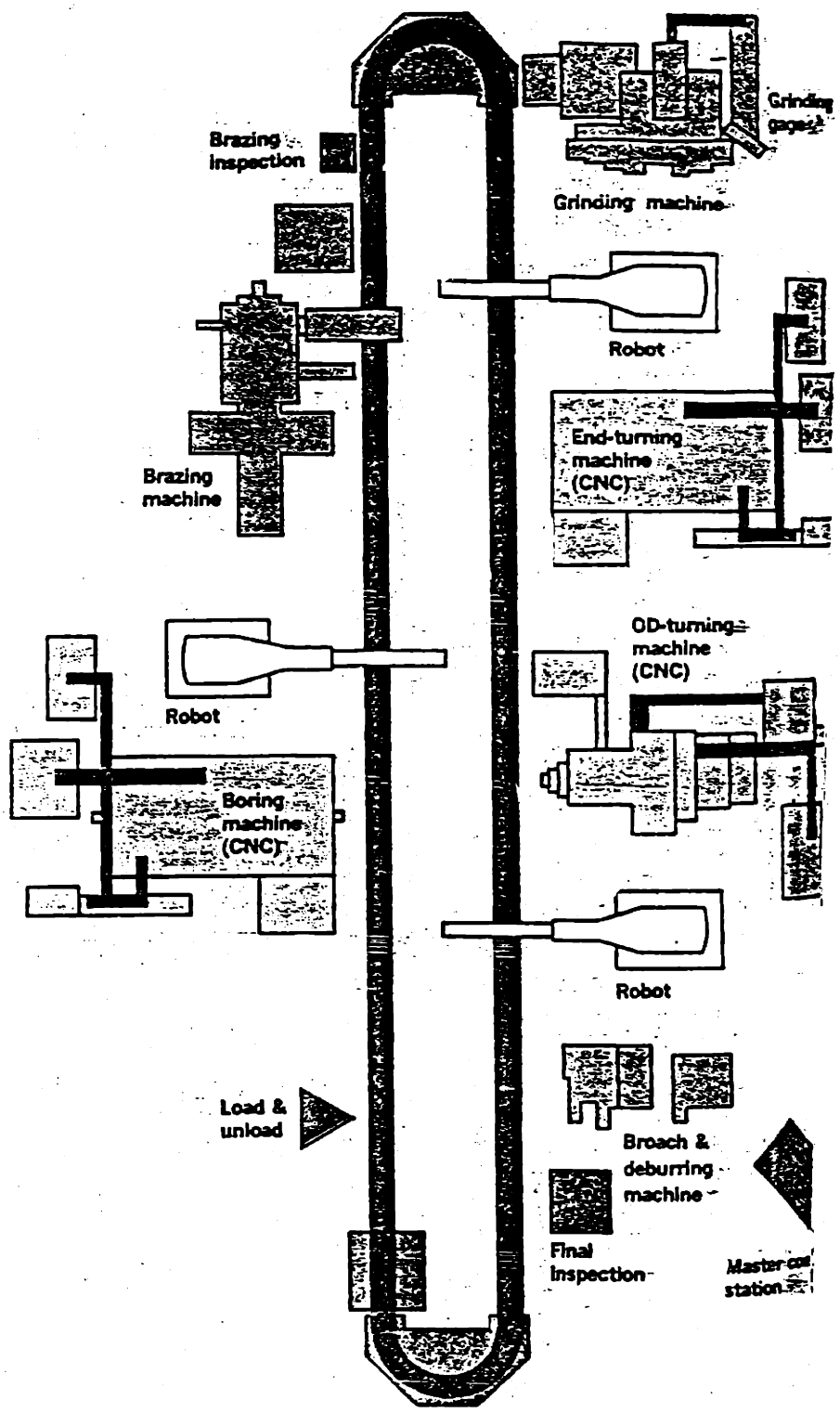

Figure 1.4 The Xerox Fuser Roll Machining Line.

In the electronics industry, flexible systems are being installed for components insertion into printed circuit boards [Gershwin et al, 1980] and integrated circuit fabrication. An interesting example of the latter is provided by the International Business Machines,Corporation system illustrated in Figure 1.5. Different ICs are produced each with a machine readable code inscribed. Control and data handling is exercised by an IBM 370/168 and nine IBM system 7 machines [Brunner et al., 1981]. The system features automated handling, storage and routing of IC wafers with the 10 computers sharing the control functions in a hierarchical fashion shown in Figure 1.6.

A system installed by Messerchmitt-Bolkow-Blohm for machining aircraft parts also exhibits a hierarchical sharing of production control and scheduling among an IBM 3032, three DEC PDP 11 units and a Siemens 330, [Lewald, 1981].

Development is taking place in the design of the elements that make up an FMS. Diagnostic software is being used to detect and identify fault conditions on machining centers [Wick, 1979]. On-line gauging and inspection may make it possible in the future to manufacture parts to high tolerances without the use of accurately made and expensive fixtures [Pressman and Williams, 1971]. This development will increase the flexibility of the workcenters. Recent developments in small machining centers are bringing improved productivity to job shops [Hartwig, 1982]. Although it is now technically feasible to integrate small versatile machining centers using robots, experts have not yet agreed on the economic feasibility of doing so [Hartwig, 1982].

1.2 Problem Definition and Outline of Thesis

The ability to produce a family of parts gives flexible workcenters significant advantages over traditional production methods. The part family

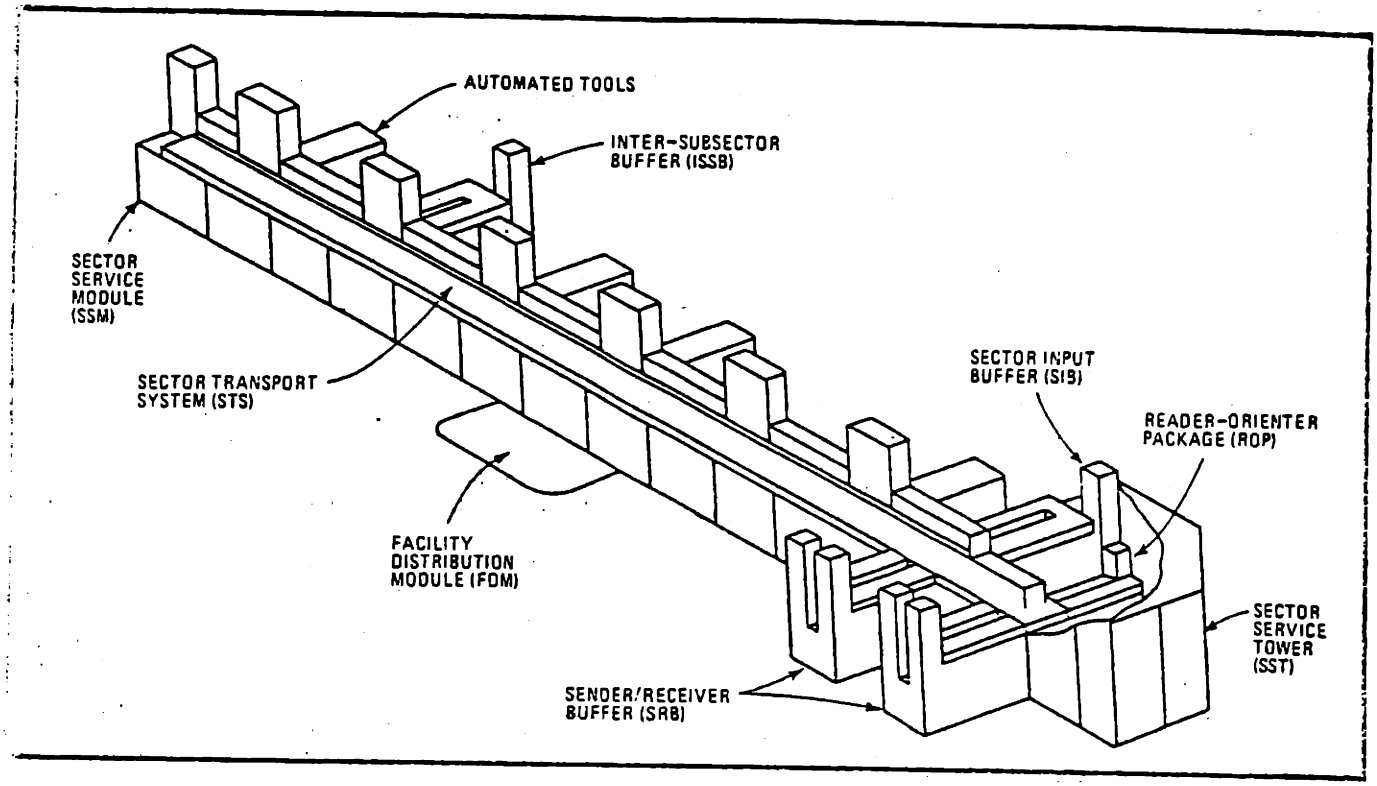

Figure 1.5 The IBM Automated Integrated Circuit Fabrication System

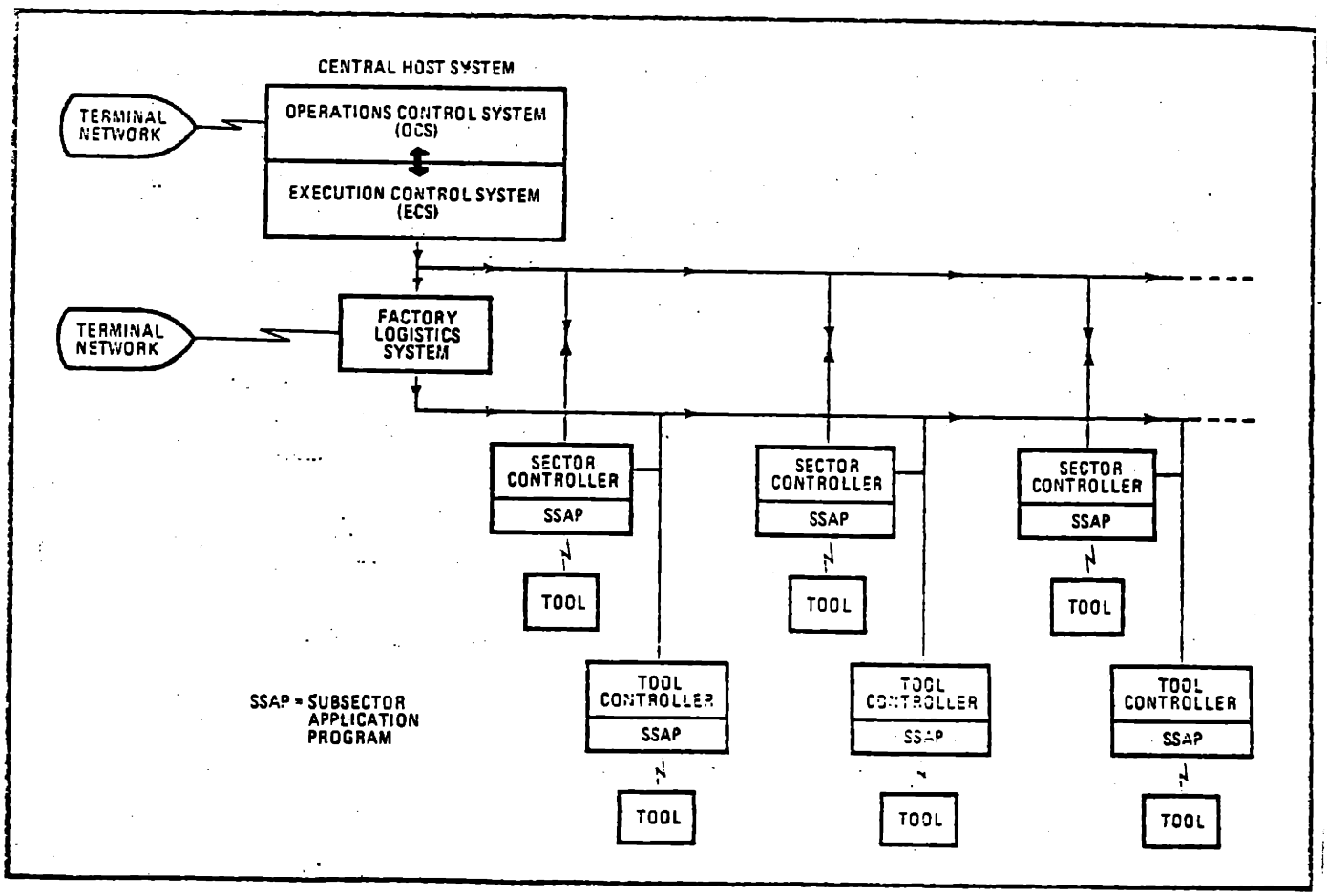

Figure 1.6 The Hierarchical Control Architecture for the IBM System

is produced simultaneously. This leads to a reduction in inventories because it is no longer necessary to produce and store batches of parts. It has been estimated that the average part in a metal cutting job shop spends approximately 3% of the time actually having useful work done on it [Sullivan, 1979]. The rest is spent waiting in storage areas. The close control afforded by an automated system should decrease dramatically the proportion of time that parts spend waiting thereby reducing the inventory of inprocess parts.

Machine tools in traditional non-automated job-shops may actually be cutting metal 5% of the time whereas in automated systems, metal cutting time may exceed 70% of the available time [Lerner, 1981]. The result is a large increase in productivity.

In order to reap full advantage from flexible automation, careful planning and control of production is necessary [Hutchinson, 1977]. This is made difficult by the fact that the workstations in an FMS are prone to failures. The time between failures and the time to repair are randomly distributed. Production planning and control algorithms must take into account the reliability of the workstations otherwise, the advantage of reduced in-process inventories offered by flexible automation may be lost.

The managers of an automated workcenter must first decide on a production plan for the part family. To do this, they should have accurate knowledge of the productive capacity of the workcenter. Because the system produces a family of parts, each with different processing times at the machines, the system capacity cannot be stated by a single number. We define the capacity as the set of all possible part mixes that can be produced

per unit time. Furthermore, because the machines in the workcenter fail and get repaired, the capacity set changes with time. The hierarchy of control computers schedule the dispatch of parts into the system and route the parts to available workstations so that the part requirement is either met or exceeded. In order to achieve this objective while keeping a small inprocess inventory, the changes in the capacity set must be taken into account.

The workcenter production planning and control problems are described in Chapter II. Workcenter managers impose part requirements on the system with the objective of satisfying a master production plan. Production control is performed by a three-level hierarchical control algorithm consisting of flow, routing and sequence control levels.

The input to the algorithm consist of the production requirement over a given time horizon, the location in the workcenter at which all operations can be performed, the time required to complete the operation at each available station and reliability data on all the workstations. The output is a schedule of dispatch times for the workpieces and the expected utilization of available time at each workstation.

A model for the flow control level of the hierarchy is presented in Chapter III. The flow control level regulates the rate at which parts enter the workcenter. It reacts to workstation failures and repairs, and to the deviation of actual production from the desired quantities. The flow control level is formulated as a stochastic optimal control problem for a system subject to sudden changes of structure due to workstation failures and repairs. We show that optimal flow control policies are piece-wise constant functions of downstream buffer levels for each failure state of the system.

In Chapter IV, we consider practical techniques for computing sub-optimal

control policies. Policies that do not rely on feedback information of the current state of production are shown to be related to existing flow optimization techniques for flexible workcenters. However, such policies may be sensitive to random disturbances and might not satisfy the demand with sufficient accuracy. Policies that make use of feedback information on both the failure state of the workstations and the current status of production are less sensitive to random disturbance. Feedback policies also satisfy the part requirements with greater accuracy than non-feedback policies.

Production planning tools for workcenter managers are developed in Chapter V. The productive capacity of the workcenter is defined by an equivalent capacity set which takes into account the reliability of the workstations. The chapter shows how the equivalent capacity set is used to determine feasible part requirements on the workcenter.

The three level hierarchical control algorithm is implemented on a simulation model of a planned flexible assembly system. Chapter VI presents the simulation results for six and sixteen machine versions of the model.

Generalizations and areas for future research are discussed in Chapter VII. Extensions to the workcenter model and improvements to computational algorithms are suggested. The problem of coordinating and controlling an integrated system with several flexible and non-flexible workcenters is also briefly described.

1.3 Literature Survey and Related Problems

Flexible Manufacturing Systems are relatively new innovations in industry and research on the subject has concentrated on the modeling and optimization of the material flow within the system [Buzacott and Yao, 1982]. It is general-

ly understood that an FMS can lead to a reduction in part inventories and give a faster response to changes in demand [Cook, 1975]. However, the impact of flexible systems on the overall production process has not been analyzed in as much detail as the internal behavior of a workcenter.

Extensive simulation studies of FMS's have been made. Simulation models have been used to plan and develop operational control rules for FMS's manufactured by The Kearney and Trecker Corporation [Hutchinson and Hughes, 1977; Hughes, 1979]. General models have been developed in order to study generic design and control problems in FMS's [Eversheim and Westkamper, 1977;Athanset al, 1977]. Simulation models can be costly in terms of computer time because many runs are required when testing for different parameter values. Recently, hybrid analytical/simulation models have been proposed which allow for more information to be determined from a single simulation experiment [Shanthikumar and Sargent, 1980; Ho and Cassandras, 1980].

Hutchinson (1977) studied the information and algorithms required for the control of a workcenter. The conclusion is that a more complex communication and control system is required compared to that needed for traditional production methods. This point was also noted by Borzcik (1980) and Nof et al., (1979).

Several analytic models of FMS's have been based on a network-of-queues approach. The workstations are modeled as service nodes in a network and the work pieces move through the network receiving service at the nodes until they leave the system at an unloading station. Network-of-queues models have been used to optimize part routing [Kimemia and Gershwin, 1980; Secco Suardo, 1979] and to study the effect of different system configurations [Nof and Solberg, 1980; Buzacott, 1982]. The criterion used to judge the FMS is the production rate of the part family. These models however, do not consider workstation

failures.

Deterministic combinatorial scheduling techniques have been suggested [Hitz, 1980; Kanellakis, 1978]. A periodic scheduling algorithm suggested by Hitz (1980), constructs a schedule for a part family "kit" consisting of the smallest integer number of parts that meets a specified part ratio constraint. The requirement on the schedule is that it should leave no idle time on bottleneck workstations. The computational cost for solving combinatorial scheduling problems is very high. This is because the computational complexity typically increases exponentially with the number of available options. Although it is possible to construct heuristic algorithms which perform reasonably well [Kanellakis, 1978], combinatorial scheduling methods do not seem feasible for production control in flexible workcenters because of their computational complexity and because they are inherently deterministic [Buzacott and Yao, 1982].

The performance of an FMS with unreliable workstations is closely coupled to the operational policies used to control the flow of material. Hildebrandt (1980) considers the problem of minimizing the time required to produce a given quantity of parts. The effect of machine breakdown is included in his model and a three-level hierarchical controller is used to schedule production. A static optimization for the first level gives part routing for all failure conditions. His work differs from ours in that the part routing does not depend on feedback information on the current state of production. Frequent solution of the static optimization problem is therefore necessary if part requirements are to be satisfied accurately. Olsder and Suri (1980) use a dynamic programming formulation for the minimum time problem. In this case, the feedback policy depends on the current failure state and production

levels.

The system of Figure 1.7 has been studied by Hahne (1981) and Tsitsiklis (1982). The problem is one of choosing the proportion of material to be sent to each downstream machine so that throughput is maximized. The optimal policy is affected by the processing times at downstream machines and by the reliability of the workstations. For each failure condition the optimal proportion of parts to send to station 1 is a piece-wise constant function of the buffer levels x^1 and x^2. The exact calculation of the optimal policy has large computational requirements [Hahne, 1981].

Hierarchical control methods have been suggested by a number of authors [Chong and Athans, 1973; Cruz, 1975]. From a practical point of view, hierarchical control schemes can be implemented in stages [Mesarovic et al, 1970]. Furthermore, subsystems can be modified without redesigning the entire control system. In manufacturing, a hierarchical management structure is common. It is important that a computer control scheme should fit into the existing organization. A hierarchical production control algorithm is therefore a natural way of converting aggregated management production plans into detailed shop floor control actions. This approach has been successfully applied in the control of steel rolling mills [Isobe, 1970].

Related areas with a large body of literature are production planning and inventory control. The problem is to make a sequence of purchasing or manufacturing decisions so as to satisfy demand while keeping the inventory of parts small. The single commodity inventory control problem has been studied extensively [Aggarwal, 1974]. A typical optimal policy is to order new stock whenever the inventory drops below a certain quantity [Bertsekas, 1976].

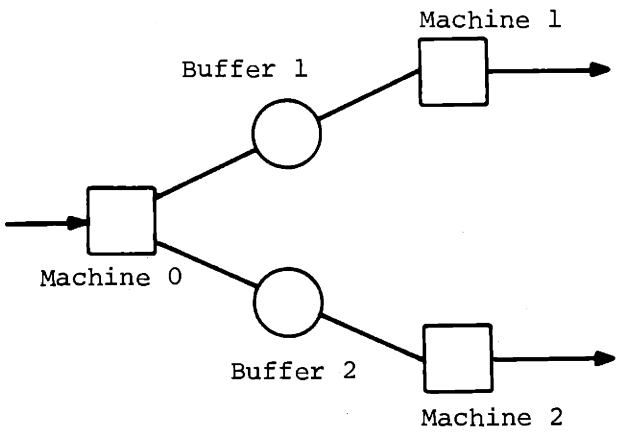

Figure 1.7 A Three Machine Diverge Network

The multicommodity inventory control problem has an order policy that depends on the vector of inventory levels [Zangwill, 1966; Veinott, 1965]. The re-order point of the single commodity problem generalizes into regions in multi-dimension space which correspond to different order quantities. Where the cost function and constraints on production are continuously differentiable, the regions have smooth boundaries [Karmarkar,1979; Deuernmeyer and Pierskalla, 1978].

Optimal control theory has been applied to production planning problems [Kleindorfer et al., 1975; Bradshaw and Erol, 1980; Hitomi and Nakamura, 1976]. Various problems studied in the literature can then be shown to be special cases of a generalized production control problem. It should be noted that in production planning, variables such as the size of the workforce, investment in new plant, amount of overtime work are often of interest. Where more than one product is made, set up costs are also included in the problem.

CHAPTER II

2.0 PRODUCTION CONTROL IN A WORKCENTER

2.1 Introduction

In the majority of implementations, flexible workcenters are part of a multi-stage manufacturing process. The input to a workcenter consists of parts that have undergone one or more processing stages. The output is a family of parts that are assembled into finished products or sub-assemblies.

The management of a manufacturing firm makes production plans for finished products by considering forecasts of demand, sales, raw material availability, inventory levels and plant capacity. From the resulting master production plan, the requirements for all the components that go into the final products can be made. The process is termed material requirements planning (MRP) [Orlicky, 1979]. It uses a component parts explosion or bill-of-materials and lead times to produce a quantity and timing requirements for the parts. The various departments responsible for the manufacture of the components schedule their activities so as to meet the requirement dictated by the master production and the materials requirement plan [Hitomi, 1979; Halevi, 1980].

In a workcenter, the operations at the workstations and the material handling system are entirely under computer control. Decisions such as which parts should be loaded into the system and what workstations particular workpieces should visit next are made by the center controller. Human intervention is necessary only when unusual events take place. It is important therefore to develop models and algorithms which allow the controller to

generate production schedules which satisfy or exceed demand requirements and to exercise control over the system so that the schedule is met. The task of the controller is complicated by the random failures of the workstations. A good production policy should anticipate failures and demand changes if it is to satisfy all of the objectives stated above.

There are two conflicting requirements on the controller. It should have a horizon of at least several hours so that it can anticipate random workstation downtimes and take into account known future changes in production requirement. At the same time, the controller must keep track of and schedule the movements of individual workpieces in the system. This is a task that needs highly detailed information about the workcenter.

In a system of reasonable size, the computational cost of a single centralized control algorithm would be immense. It is proposed therefore to divide the production control problem into a hierarchy of smaller manageable sub-problems. Each level of control is characterized by the length of its planning horizon and the type of information needed. Higher levels in the hierarchy typically have long horizons and use highly aggregated data while lower levels have short horizons and use more detailed information. The nature of the sources of uncertainties at each level of control also differs.

A three level hierarchy specifically designed to compensate for workstation failures is proposed. The hierarchy is illustrated in Figure 2.1, in which the workcenter controller is embedded in the larger hierarchy of production management.

The master production and requirements plan specify production targets over periods one to several weeks in advance. The workcenter managers determine the tooling at the workstations and daily production targets for the

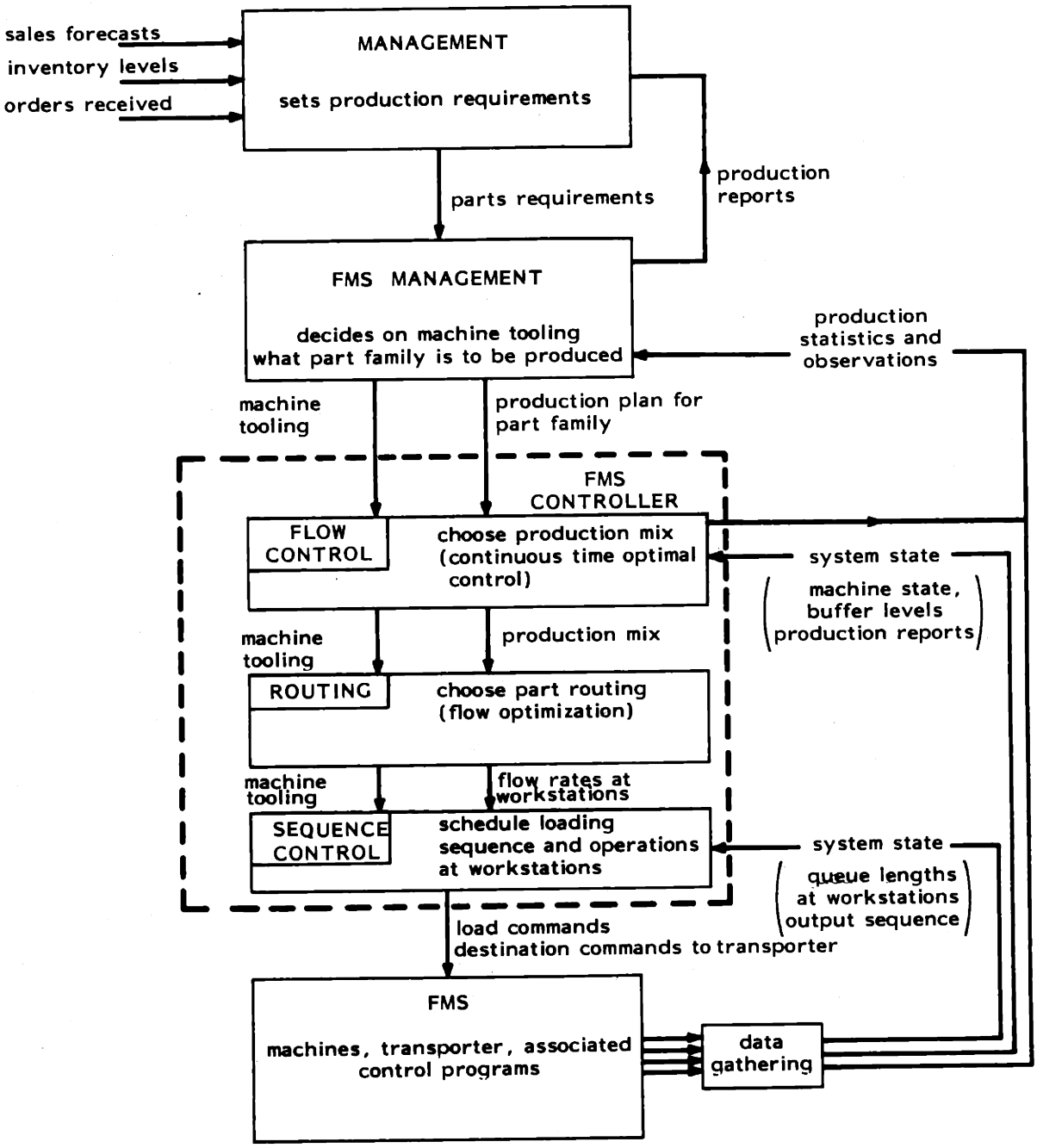

Figure 2.1 The Hierarchical Production Control Scheme

part families to be produced with the objective of meeting the master plan [Hutchinson, 1977; Hitomi, 1980]. The workcenter controller has the task of scheduling the operation of the manufacturing system so that the daily production targets are met. It should be pointed out that the length of planning horizons given above only indicate relative magnitudes. Actual lengths depend on the type of manufacturing system involved. For example, metal cutting workcenters have been designed to operate unmanned for the duration of a single night shift [Lerner, 1981; Skole, 1979]. As flexible automation becomes more widespread, the length of economic production runs is likely to become shorter than the current norm [The Economist, 1981].

The hierarchical workcenter controller makes decisions based on up-to-date information about the system. It is therefore a closed loop or feedback control system. Feedback control schemes are less sensitive to modelling errors and random disturbances than open loop control policies. However, closed loop policies can be computationally expensive because the decision rules often require a great deal of off-and on-line computation and storage. In manufacturing systems this is not a great disadvantage because events such as operation completions, failures or repairs happen slowly compared to the computational speed of modern computers. The size of the data base required to operate an automated manufacturing systems means that fairly large computer resources are installed and available for calculating control policies.

The top level of the controller hierarchy is the flow control algorithm which is described in Section 2.2. Its task is to continuously regulate the production rate of each member of the part family. The flow controller takes into account known future changes in demand and statistical information on work-

station reliability. The flow controller planning horizon may therefore be several hours to days in length.

Once the production rates have been set the second level routing controller, described in Section 2.3, determines the routes that the parts should follow through the system. Section 2.4 discusses the bottom level sequence control algorithm which dispatches individual pieces through the workcenter. The routing and sequence controllers are activated each time the flow control algorithm changes the throughput rates. This may happen as often as once every few minutes depending on the number of workstations, and their failure and repair rates.

Practical considerations in implementing the workcenter controller are discussed in Section 2.5. These are the limitations imposed by the assumptions made in deriving the hierarchical controller. The interactions between plant personnel and the control algorithms are also considered. These involve additional estimation and predictive algorithms which monitor the performance of the system and alert workcenter personnel when unusual events that might adversely affect production occur. These algorithms together with the off-line calculations to determine production policies can be considered as an additional higher level of the control hierarchy.

There are other kinds of uncertainties such as inspection, part rejection and material unavailability that are not yet incorporated in the workcenter model. Such events fit into the framework of our approach. Chapter VII suggests possible ways of incorporating additional sources of random disturbances into the workcenter model.

2.2 Flow Control

The flow control algorithm adjusts the part mix being produced by the workcenter so that the targets set by the managers of the facility are attained. The flow regulation must be done continuously so as to respond to the random failures and repairs at the workstations. If, for example, a part cannot be produced when a workstation has failed, the lost production must be made up when that machine is operational. Using failure and repair statistics, the production rates are chosen in a way that anticipates station down times. Adequate but not excessive inventories of finished parts are maintained so as to hedge against machine failures.

The flow control model is shown in Figure 2.2. The part flow is modelled as a continuous process. The workcenter is modelled as a processing system whose state depends on the set of operational workstations. Corresponding to each state is a capacity which is the set of the throughput rates that can be achieved with the set of operational workstations. The capacity of a workcenter is thus a set that varies with time. The workcenter controller should therefore have information on the extent of the capacity set at all times.

The downstream buffers hold finished workpieces and serve to decouple the workcenter from subsequent production stages.

We assume that the times between failures and repairs or, in other words, between state changes of the flow control model are long compared to the time it takes to produce a part. It is important that the rates at which parts enter the system should be within the capacity of the set of operational stations. Failure to comply with this constraint leads to a buildup of in process parts within the system, resulting in congestion on the transportation

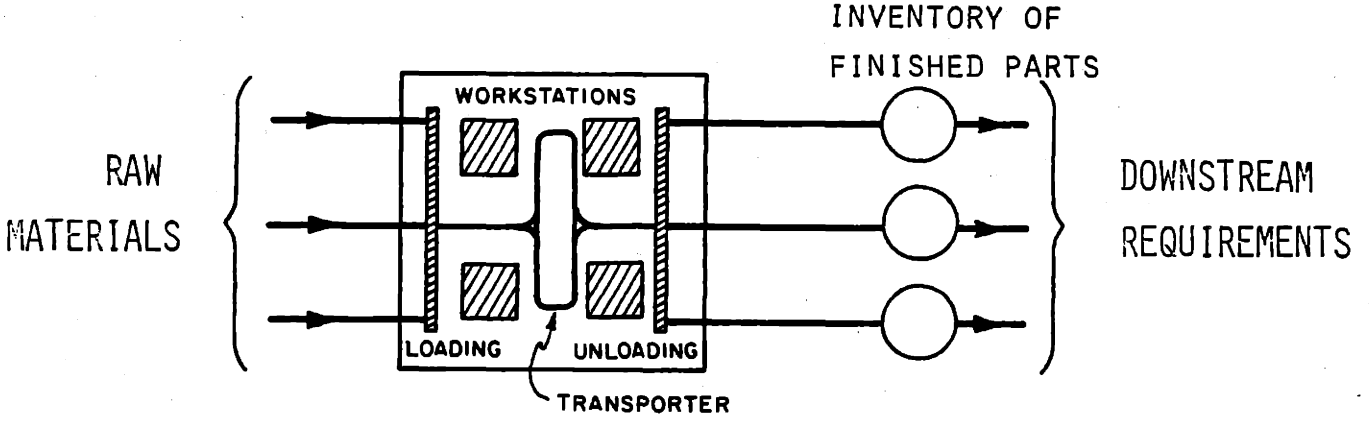

Figure 2.2 The Flow Control Workcenter Model

system and blocked stations. These effects may actually lower the productive capacity of the system.

The flow control problem is formulated as a stochastic dynamic control problem in Section 3.2. Application of the optimality conditions of dynamic programming reveals that the optimal policy is a feedback control law which is a function of the machine state and downstream buffer levels. Furthermore, as a function of time, the optimal production policies are piece-wise constant except for certain buffer states when they may be time varying. In other words, the flow control algorithm tends to maintain constant throughput rates over short intervals of time. The length of the intervals depend on the machine state, the demand rate, and the failure and repair rates of the workstations.

The feedback policy is calculated off-line and stored. On-line, the controller uses the stored decision tables to select the production mix for the current machine state and downstream buffer levels. The on-line selection may be performed at regular intervals of 1 to 5 minutes between machine state changes and immediately whenever a failure or repair occurs. In most invocations, the flow control algorithm has only to verify that the current production rates are still optimal because of the piece wise continuity of the control policy.

The off-line calculation gives the expected deviation of production from the requirements. This prediction is useful to the managers of the workcenter in their production planning process. The interaction between the flow control algorithm and the plant personnel is important and is discussed in more detail in Section 2.5.

The stochastic dynamic production control approach results in a feedback policy that takes into account statistical information on workstation reliability and the known demand requirements. The control policy therefore responds to failures and repairs as they occur, hedges against future station down-time and utilizes all available capacity.

2.3 The Routing Algorithm

A part in going through a flexible manufacturing system often has a choice of two or more stations for some of the operations it requires. The middle level routing algorithm determines the proportion of parts that should go to each station whenever such a choice is available. More generally, it chooses among available paths for each part.

The manufacturing system is modelled as a network-of-queues with the stations being the service nodes and internal storage buffers holding the queue of waiting parts. The arrival rate of parts into the system is the production rate set by the flow control algorithm. Since the rate set by the flow controller is guaranteed to be within the capacity of the system, it is sufficient for the routing controller to pick feasible proportions for the operations for all of the parts. Network of queues analysis techniques can then be used to determine the bottleneck stations and average times required to completed each part and the average number of pieces in internal storage buffers [Solberg, 1977].

In many cases, the routing for a given production rate may not be unique. When this is the case, a routing that minimizes some performance criterion can be employed. Network-of-queues models show that for a wide class of service time distributions, the marginal probability of having a given number

of pieces at a certain station, when the throughput is fixed, depends on the mean service time and the mean flow rate through the station [Basket et al, 1975]. By varying the proportion of parts that have a certain operation performed at a given station, then mean flow rates and service times at the station can be affected. The routing algorithm thus has some control on the average number of pieces in the system.

For a given throughput, the routing that minimizes the average number of pieces within the workcenter is to be preferred. Keeping the number of inprocess parts small reduces congestion on the transportation mechanism, and where internal storage inside the system is limited, reduces the incidence of blocked workstations. In certain types of manufacturing systems, the parts travel on special fixtures or pallets. Reducing the number of inprocess parts means that fewer pallets are required in order to maintain the throughput, or conversely, that greater production is possible with a given number of pallets.

A number of techniques are available for optimizing the routing of parts through a workcenter [Kimemia and Gershwin, 1980; Secco Suardo, 1981; Buzacott, 1982]. An appropriate method can be incorporated in the routing algorithm when the hierarchical controller is implemented.

2.4 The Sequence Controller

The bottom level sequence controller makes two major decisions. It determines the sequence and time that parts should enter the system. When a piece completes an operation at a workstation, it decides what operation, and at which station, should be done next. Unlike the flow and routing algorithms, the sequence controller deals with individual work pieces, and

interacts directly with the software driving the system. The information required at this level of control is the status of every component of the workcenter and the location of every piece in the system.

The objective of the sequence controller is to maintain the throughput set by the flow controller and the operation splits determined by the routing algorithm. The task is one of establishing feasible schedules or establishing decision rules so that the objective is satisfied.

In simulation studies, we have found a fairly simple interval loading scheme to be effective in maintaining the flow rates. The production rate for part n is u^n and is determined by the flow control algorithm. The sequence controller establishes for each part, windows of length $1/u^n$ starting at the time t_0 that the rates u^n are set. The windows give the earliest and latest times that a part can be loaded. Thus the earliest loading time for the ℓth type n part is $t_0 + (\ell-1)/u^n$ and the latest time is $t_0 + \ell/u^n$. If a part can be loaded in each window, then the flow rates are maintained.

The simplest implementation of the algorithm tries to load a part as early as possible in the window, but a more sophisticated approach is to look ahead and select the times at which the part is loaded within the window so that machine blockage and starvation is minimized. Because the production rates set by the flow controller are within the capacity of the system, the interval loading scheme is guaranteed not to introduce material into the system at a rate that cannot be processed. The inprocess inventory is thereby kept low.

The routing algorithm determines the splits for each part whenever an operation can be performed at more than one station. The sequence controller uses the flow rates and the splits to make decisions for individual pieces on which operation should be performed next and at which station.

The interval loading scheme is easy to calculate and to implement. It is also a feedback control scheme because it uses current system status information to make scheduling decisions. One further advantage, is that it can be implemented in a decentralized manner. A feature common to some of the manufacturing systems described in Section 1.2 is a control architecture with several different computers taking charge of different sectors and functions in the system. If the throughput rates and operational splits selected by the flow and routing control algorithms are made available to all the sector controllers, they can then make decisions on the movement of parts within their sectors independently.

2.5 Practical Considerations in the Implementation of the Hierarchical Controller

Feedback control policies give the hierarchical controller a certain degree of robustness with respect to modelling errors and random disturbances. A number of assumptions are made in deriving the models used to develop the control algorithms. An understanding of the effects of violating the assumptions is necessary before the hierarchical controller can be implemented.

The flow control model assumes that the chosen production rates can be maintained by the routing and sequencing algorithms. The simulation results of Chapter VI and those reported by Kimemia and Gershwin [1980] show that throughput rates can be maintained with low inprocess inventories if they are within the capacity of the system. The capacity depends on the processing times of the parts at the workstations and the reliability of the system. However, only estimated values of the mean time between failures and repairs can be used in production planning. These parameters are not known exactly and what is more, they may vary over time.

It is possible that a calculation with inaccurate parameter values will show a demand requirement on the system to be feasible when in fact it is not. Using those parameter estimates, the flow control algorithm will then set production rates which the sequence controller cannot achieve. On the other hand, if the capacity is under estimated, the throughput does not fully utilize the workstations. In either case, the control system is perceived as operating poorly.

In a practical implementation, a method of keeping track of system parameters, especially failure and repair rates, is necessary. When discrepancies between observed and apriori estimates occur, a decision is made whether or not to recalculate the decision tables on which the control policy is based. If in the light of observed parameter values it is determined that demand can no longer be satisfied, the managers of the workcenter can be alerted so that they can take appropriate action.

The overall production planning and control scheme is shown in Figure 2.3. Workcenter production planners base their plans on the best current estimates of the system parameters. They also have access to information on the system and predictive models which allow them to forecast the behavior of the system. Chapter V develops a dynamic model which can be used for the production planning problem.

The off-line calculations of the flow controller generate decision tables which are used on-line to calculate the production rates. These decision tables are available to the predictive models. The off-line calculations are fairly expensive and a decision to recalculate them should only be made when the predictive algorithms indicate that a deterioration in performance would otherwise occur. Up to date parameter estimates can be

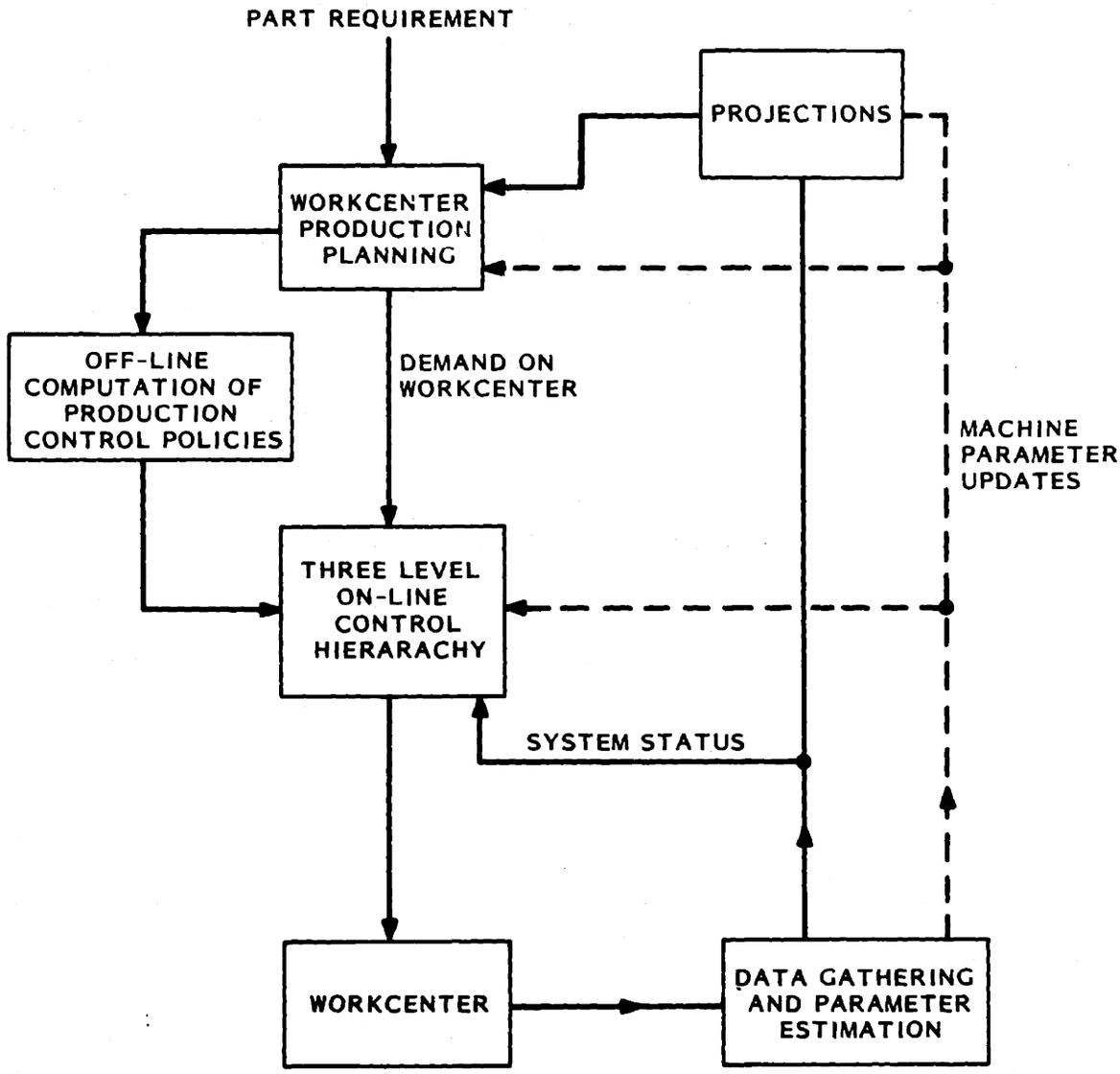

Figure 2.3 Overall Production Planning and Control System

utilized for on-line calculations.

The sequence control algorithm interacts with the software that drives the workcenter. It must be robust with respect to phenomena not modelled in the flow control model. In an operating system there are random disturbances apart from the failures and repairs of the workstations. The chosen throughput rates should be maintained despite unanticipated disturbances. Feedback policies enable the sequence controller to maintain the production rates so long as the disturbances do not reduce the capacity of the system. Random effects that reduce the capacity should be modelled. This may give additional parameters to be estimated.

2.6 Summary

This Chapter has given a qualitative description of an approach to the production control problem facing the managers of a flexible automated manufacturing or assembly system. A dynamic feedback control algorithm is described which is designed to track production targets which may be changing over time.

The controller fits into existing management structures and also serves as part of the interface between plant personnel and the production hardware and software.

In the next Chapter, we give the mathematical models of the system, a technical discussion of the hierarchical controller and characterize optimal production policies.

CHAPTER III

3.0 FLOW CONTROL POLICIES FOR THE AUTOMATED WORKCENTER

3.1 Introduction

The hierarchical production control algorithm is responsible for meeting production targets set by the managers of the facility. In this chapter, we examine in detail the model for the flow control level of the controller hierarchy and derive characteristics of the optimal flow control policy.

Section 3.2 introduces the model and formulates the stochastic flow control problem. The random elements to be studied are the failures and repairs of the workstations in the system. The stations are assumed to fail and be repaired independently. The times between failures and repairs are modelled by independent exponentially distributed random variables. The mean time to repair and to fail are properties of each machine which are assumed not to depend on the operations being performed. These parameters remain constant over the horizon of the hierarchical controller. The failure and repair processes are therefore memoryless and can be modelled by a Markov process.

The assumption of memoryless failures and repairs is common in the literature [Buzacott,1967]. It is suitable where there are many possible causes of failures such as tool breakages, workpiece jams and electrical faults. In automated manufacturing systems, where the machines are numerically controlled, software crashes are additional causes of failures. Samples of actual up and down time distributions reported by Shick and Gershwin, [1978] are in fact close to exponential.

Some causes of downtime do not satisfy the memoryless assumption. These include scheduled down times because they are predictable from the history of the system and systems in which there are a limited number of repair personnel because the length of a down time when several machines have failed may depend on the order in which the failures occured.

Scheduled maintenance can be lumped together with other failures and treated as though it was an unforseen station down time. Alternatively, planned down time can be modelled as a set of additional parts which require an operation at each of the workstations with a duration equal to the maintenance period. The hierarchical controller then does the useful additional job of scheduling maintenance for the workstations.

A limited number of service personnel can be represented by a memoryless model if it is assumed that they distribute their work equally amongst failed machines. The effect of this assumption is to increase the expected time to repair when the number of failed machines exceeds the number of repair personnel. This in turn has the effect of lowering the effective overall capacity of the workcenter. It also makes the flow controller maintain higher downstream buffer levels as a hedge against failures. As a result, we expect that the controller's performance will not be degraded by a memoryless model when the size of the repair crew is limited.

The optimal control policy for the finite horizon flow control problem is derived in Section 3.3. The finite horizon formulation is suitable for production control problems with time varying demand and in which the time to produce a part is of comparable length to the horizon over which the flow control policy is calculated. For example, in metal cutting, a part may take

up to 3 hours to complete all operations and the horizon is of the order of 8-16 hours.

The flow controller's objective is to minimize a cost index which is a measure of performance. Typically, we penalize the controller for deviations of actual production from the desired values.

Whenever the demand in a flexible production system is constant over time periods that are long compared to the time taken to produce individual parts and, the mean time to fail and repair, the production planning horizon for the flow controller may be effectively treated as being infinite.

In the infinite horizon production control problem, there are two objectives that we could seek to minimize. First, there is the total discounted cost index. In this case the immediate cost in terms of failing to meet the demand or carrying an excessive number of finished parts is more important than the future cost. Second, when the long term performance of the system is of interest, the average value of the cost index per unit time can be minimized.

The infinite time discounted cost problem is discussed in Section 3.4. Under fairly mild conditions, steady-state optimal controls exist.

The usefulness of the discounted cost index is limited by the fact that for a given control law, a finite cost may exist even though the buffer state $x(t)$ becomes unbounded. This behavior depends on the particular cost function and the linear dynamics of the buffer state.

The average cost problem discussed in Section 3.5 presents technical difficulties because the buffer state $x(t)$ may become unbounded as t goes to infinity. At the heart of the problem is the existence of control policies which can meet the given constant demand. In other words, the existence of

control laws for which the buffer state x(t) is stable, in some sense, must be shown. This is of particular interest to managers since they would like to know in advance whether or not their manufacturing plant can indeed meet the demand requirement. We overcome the technical difficulties by imposing bounds on the buffer state and requiring that for any admissible control policy, long term production rate equals the demand rate.

3.2 The Model for the Flow Controller

The workcenter consists of M workstations on which N parts are produced. The material flow is modeled as a continuous process. Let $u(t)\varepsilon R^N$ be the instantaneous production rate vector of the system, the control variable. The downstream demand rate is $d(t)\varepsilon R^N$ and is known over the interval (0,T). Finished parts are stored in a buffer from where the downstream requirements are satisfied. Define $x(t)\varepsilon R^N$ by the following differential equation

$$\frac{dx(t)}{dt} = u(t) - d(t). \tag{3.1}$$

The vector x(t), termed the buffer state, measures the cumulative difference between production and demand for the part family. A negative value for a component of x(t) gives the backlogged demand for the corresponding part. A positive value is the size of the inventory stored in the downstream buffers. Ideally, parts in a workcenter should be produced as they are required, keeping the buffer state x(t) zero.

The state of the workstations is described as the machine state and is denoted by an M-tuple of binary variables $\alpha(t)$ with the mth component defined by

$$\alpha_m(t) = \begin{cases} 1 \text{ if station m is operational} \\ \\ 0 \text{ if it has failed} \end{cases}$$

Given that workstation m is operational, the probability of a failure in an interval of length δt is assumed to be $p_m \delta t + 0(\delta t^2)$. When station m has failed, the probability that a repair is completed in the interval $(t,t+\delta t)$ is assumed to be $r_m \delta t + 0(\delta t^2)$. The parameters p_m and r_m are the failure and repair rates of station m. The dynamics of $\alpha_m(t)$ are therefore governed by

$$P(\alpha_m(t+\delta t) = 1 \mid \alpha_m(t) = 0) = r_m \delta t + 0(\delta t^2) \tag{3.2}$$

$$P(\alpha_m(t+\delta t) = 0 \mid \alpha_m(t) = 1) = p_m \delta t + 0(\delta t^2) \tag{3.3}$$

The time between failures and the time to repair are thus modelled by exponentially distributed random variables with means $1/p_m$ and $1/r_m$ respectively. The machine state can be modelled by an irreducible Markov chain with 2^M states. Each state communicates with M neighbors and transitions are due to the failure or repair of a single station.

It is convenient to represent this behavior with more abstract notation. Let S be an index set corresponding to the machine states. Then for $i,j \in S$

$$\left.\begin{aligned} P(\alpha(t+\delta t) = j \mid \alpha(t) = i) &= \lambda_{ij}\delta t + 0(\delta t^2) \quad , i \neq j \\ P(\alpha(t+\delta t) = i \mid \alpha(t) = i) &= 1+\lambda_{ii}\delta t + 0(\delta t^2) \end{aligned}\right\} \tag{3.4}$$

The transition rates λ_{ij} are functions of the failure and repair rates p_m and r_m. Note that

$$\lambda_{ij} \geq 0, \ \lambda_{ii} = -\sum_{j \neq i} \lambda_{ij}$$

The model assumes that a machine's failure rate does not depend on the part flow through the system. Where reliability depends on the parts being processed, the failure rate becomes a function of the part production rate and the routing of parts within the workcenter. In this case, the routing must be determined simultaneously with the production rates. The production rate vector u(t) is constrained to lie within the capacity set $\Omega(\alpha(t))$ of the currently operational workstations. The capacity set depends on the machine state $\alpha(t)$ and is therefore subject to sudden changes.

To define the capacity set of the workcenter, consider the machine state $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_M)$. Let $w_{nm}^k \geq 0$ be the rate at which type n parts are sent to station m for operation k. Let τ_{nm}^k be the time required to complete the operation. Since it is assumed that no material is accumulated within the system, the number of type n parts undergoing operation k is equal to the throughput u^n for the part. This is expressed as

$$\sum_m w_{nm}^k = u^n \qquad \forall\ n,k \tag{3.5}$$

The limited capacity of the station is expressed as [Kimemia and Gershwin, 1980]

$$\sum_n \sum_k w_{nm}^k \tau_{nm}^k \leq \alpha_m \tag{3.6}$$

From equation (3.6), if station m has failed, then $\alpha_m = 0$. Since the flow rates w_{nm}^k are non-negative, they must necessarily be zero to satisfy (3.6). Thus the routing algorithm does not send parts to failed stations. For $\alpha_m = 1$, the value of the lefthand side of (3.6) is the proportion of time that station

m is busy because it sums the product of the number of parts per unit time by the lengths of the operations. Bottleneck stations are characterized by the satisfaction of (3.6) as an equality. The production constraint set $\Omega(\alpha)$ for machine state α is defined to be the set of all $u \geq 0$ for which there exist $w_{nm}^k \geq 0$ satisfying (3.5), (3.6) and in addition

$$w_{nm}^k = 0 \quad \text{if station m cannot perform operation k on part n} \tag{3.7}$$

The variables w_{nm}^k define the routing within the workcenter. For a given production rate vector u, the routing may not be unique. Once $u(t) \in \Omega(\alpha(t))$ is chosen by the flow controller, the part routing can be determined by the routing algorithm. The sequence controller then determines schedules based on the chosen values of u and w_{nm}^k.

The control constraint set $\Omega(\alpha)$ is the projection of the set of feasible w_{nm}^k into R^N via (3.5). From (3.5) - (3.7), $\Omega(\alpha)$ is a convex polyhedral set lying entirely in the positive orthant and containing the origin.

The flow control problem can now be stated. Given a workcenter as described above, an initial buffer state x(0) and machine state $\alpha(0)$, we wish to specify a production plan $u(t) \in \Omega(\alpha(t))$ for $0 \leq t \leq T$ that minimizes the performance index

$$J_0^i(x) = E\left\{ \int_0^T g(x(t))dt \,\middle|\, x(0) = x,\ \alpha(0) = i \right\} \tag{3.8}$$

Subject to (3.1) and (3.4). The function g(x) penalizes the controller for failing to meet demand and for keeping an inventory of parts in the downstream buffer.

It is given by

$$g(x) = \sum_{n=1}^{N} g_n(x^n) \tag{3.9}$$

where $g_n(x^n)$ are convex scalar functions satisfying

$$\lim_{|t| \to \infty} g_n(t) = \infty . \tag{3.10}$$

The performance index (3.8) is thus the expected total cost over the period (0,T) given the initial conditions x and i.

The class of controls U to be considered consists of the set of functions $u: R^N x S x R \to R^N$ that satisfy

$$u(x,i,t) \in \Omega(i) \quad \text{for } u \in U,\ i \in S,\ x \in R^N \text{ and } t \in (0,T) \tag{3.11}$$

In addition, there are continuity requirements which are described below.

In order to characterize the optimal policy and to be able to compare the cost index associated with two different control policies, we have to restrict the set of admissible policies. For the workcenter control problem, the restrictions we can apply are those that eliminate obviously undesirable or impractical control policies. We can then be confident that an optimal control law within the admissible set is indeed optimal in a wider class of control policies.

We define the admissible control set U_A to be all functions $u: R^N x S x R \to \Omega(\alpha)$ such that for all $i \in S$, discontinuities of u(x,i,t) divide R^N x R into a finite set of regions within which u(x,i,t) is a continuous function of x and t.

Furthermore, the boundary separating two regions should be continuous and piecewise smooth.

Intuitively, whenever the buffer state crosses a discontinuity in the control policy, the control vector $u(x,i,t)$ undergoes a finite jump. Physically, there is a limit to the number of such jumps in the control vector that can occur in a finite time. The limitation on the possible discontinuities in admissible control policies eliminates control laws with an infinite number of discontinuities along a buffer state trajectory. Piecewise smooth boundaries again ensure that any trajectory which moves along a boundary also experiences a control vector with a finite number of discontinuities in any finite time interval.

The admissible class of control policies consists of feedback laws which determine the production rate $u(x(t), \alpha(t),t)$ at time t for each buffer level and machine state. Once a control law is specified, equation (3.1) and (3.4) define a unique Markov process [Tsitsiklis, 1982]. In what follows, we identify $u \in U_A$ with the policy and $u(x,i,t)$ with values taken by the control vector. For a fixed control policy u and initial conditions $x(0) = x$ and $\alpha(0) = i$, we associate $[x_u(t), \alpha(t)]$ with the resulting random process. A fixed point in the state space is denoted by (x,i) with $x \in R^N$ and $i \in S$.

3.3 Characterization of the Optimal Policy for the Finite Horizon Control Problem

Define the expected cost when the policy u is applied in the interval (t,T) as

$$J_u(x,i,t) = E\left\{\int_t^T g(x_u(s))ds \mid x(t) = x, \alpha(t) = i\right\} \qquad (3.12)$$

The function $J_u(x,i,t)$ is the expected value of the cost index given that the system has a downstream buffer level x and machine state i at time t. If the machine state remains unchanged until time T, the buffer state follows a deterministic trajectory $x_u^i(s;x,t)$ which is a solution to the differential equation

$$\dot{x}(s) = u(x,i,s) - d(s) \tag{3.13}$$

with initial condition $x(t) = x$. We refer to $x_u^i(s;x,t)$ as a trajectory of x corresponding to the control law u and machine state i. The cost-to-go function (3.12) statifies, for $i \in S$ and $x \in R^N$, [Rishel, 1975].

$$J_u(x,i,t) = \int_t^T \left\{ g(x_u^i(s;x,t)) + \sum_{j \in S} \lambda_{ij} J_u(x_u^i(s;x,t),j,s) \right\} ds \tag{3.14}$$

with $J_u(x_u^i(T;x,t),i,T) = 0$ for all $i \in S$. The integration in (3.14) is performed along the deterministic trajectory $x_u^i(s;x,t)$. The first part of the integrand is the cost if the policy u is applied and there are no machine state changes until the final time T. The second part is the added cost due to the possibility of transitions to other machine states.

From (3.13) and (3.14), it follows that $x_u^i(s;x,t)$ and consequently $J_u(x_u^i(s;x,t),i,s)$ are continuous functions of s. For $u \in U_A$, $J_u(x,i,t)$ is continuously differentiable within certain subsets of $R^N \times R$. The following differential equation is then satisfied [Rishel, 1975].

$$g(x) + \frac{\partial}{\partial t} J_u(x,i,t) + \frac{\partial}{\partial x} J_u(x,i,t) \; [u(x,i,t) - d(t)]$$

$$+ \sum_{j \in S} \lambda_{ij} J_u(x,j,t) = 0 \tag{3.15}$$

in addition, an optimal policy u* and associated cost $J_{u^*}(x,i,t)$ satisfies

$$\min_{u\in\Omega(i)} \left\{ g(x) + \frac{\partial}{\partial t} J_{u^*}(x,i,t) + \frac{\partial}{\partial x} J_{u^*}(x,i,t)\ [u - d(t)] \right.$$

$$\left. + \sum_{j\in S} \lambda_{ij}\ J_{u^*}(x,j,t) \right\} = 0 \qquad (3.16)$$

An outline of the proof of (3.15) and (3.16) is given in Appendix I.

The partial differential equation (3.16) is linear in u and the control constraint set $\Omega(i)$ is polyhedral. Consequently, whenever the derivative $\frac{\partial}{\partial x} J_{u^*}(x,i,t)$ exists, the optimal control $u^*(x,i,t)$ lies at an extreme point of $\Omega(i)$. For each machine state and time t, the optimal policy therefore divides the buffer state into a set of regions within which the control is constant. Whenever the buffer state is in one of these regions, the optimal production rate vector is at an extreme point of $\Omega(i)$. However, the regions do not cover the whole space. If the derivative $\frac{\partial}{\partial x} J_{u^*}(x,i,t)$ does not exist, is orthorgonal to a face of $\Omega(i)$, or is zero over a measurable subset of R^N, then there is no unique minimizing value to (3.16). The optimal policy then depends on the extreme point policies in the regions neighboring the subset. This behavior is similar to singular control in deterministic optimal control problems.

The optimal cost-to-go function has the following useful properties

a) $J_{u^*}(x,i,t)$ is a continuous function of x and t for all $i \in S$

b) $J_{u^*}(x,i,t)$ is a convex function of x for $t \in (0,T)$ and $i \in S$.

The proofs of (a) and (b) are given in Appendix I. Continuity of $J_{u^*}(x,i,t)$ means that small changes in initial conditions do not result in large changes in the cost index. Intuitively we would expect this in practice. Convexity of $J_{u^*}(x,i,t)$ follows from the convexity of the cost function $g(x)$ and the linear buffer state dynamics. It means that the regions corresponding to control at each extreme point of $\Omega(i)$ are connected [Tsitsiklis, 1982]. Strict convexity implies that the boundaries between two regions are on lower dimensional manifolds which intersect at the minimum with respect to x of $J_{u^*}(x,i,t)$.

From (3.20), the optimal control $u^*(x,i,t)$ points in the direction of steepest descent of the cost-to-go function at the point x within the set $\Omega(i)$. If $u^*(x,i,t) > d(t)$ componentwise, then the trajectory $x^i_{u^*}(s;x,t)$ moves in a direction that decreases the value of $J_{u^*}(x^i_{u^*}(s;x,t),i,s)$ with respect to (x,s). In machine states where the demand $d(t)$ is within the capacity of the available workstations, that is $d(t) \in \Omega(\alpha(t))$, the hierarchical controller tends to drive the machine state to the point $x^*_i(t)$ defined as

$$x^*_i(t) = \operatorname*{argmin}_x J_{u^*}(x,i,t)$$

In the neighborhood of $x^*_i(t)$, the optimal trajectory points towards $x^*_i(t)$. Thus whenever the buffer state is at the minimum of $J_{u^*}(x,i,t)$, the flow control algorithm produces at a rate sufficient to keep the buffer state at that point. The quantity $x^*_i(t)$ is thus called the <u>hedging point</u> for machine state i. A similar type of behavior has been demonstrated for certain capacitated deterministic production and inventory control problems [Deuermeyer and Pierskalla, 1978].

In machine states where the demand is outside the control constraint set, there is insufficient capacity to drive the buffer state trajectory in the descent direction of $J_{u^*}(x,i,t)$. In these machine states, the optimal hedging point cannot be attained. However, it is possible that the optimal policy will build up the downstream buffer level of some pieces at the expense of others. This behavior depends on the control $u^*(x,i,t)$, buffer level $x(t)$ and the value of the cost-to-go function $J_{u^*}(x,i,t)$.

As an example, consider the single machine system of Figure 3.1 producing two types of parts. The control constraint set $\Omega(1)$ when the machine is operational is shown in Figure 3.2. When the machine has failed the set $\Omega(0)$ consists of the single point at the origin. The downstream buffer regions corresponding to the control policy when $\alpha=1$ are shown in Figure 3.3. The vertex corresponding to each region is labeled in Figure 3.2. When $\alpha=0$, no production is possible, hence there is no counterpart to Figure 3.3.

A trajectory $x^i_{u^*}(s;x,t)$ of the buffer state moves under the action of a constant control until it either hits a region boundary or there is a machine state change. In the former case, the trajectory may either cross into a neighboring region in which case the control switches to a neighboring extreme point or it may stay in the boundary region under the action of a time varying control. Control policies in the neighborhood of the boundary are examined in more detail in Appendix I. When a failure or repair occurs, the control is determined by the regions corresponding to the new machine state.

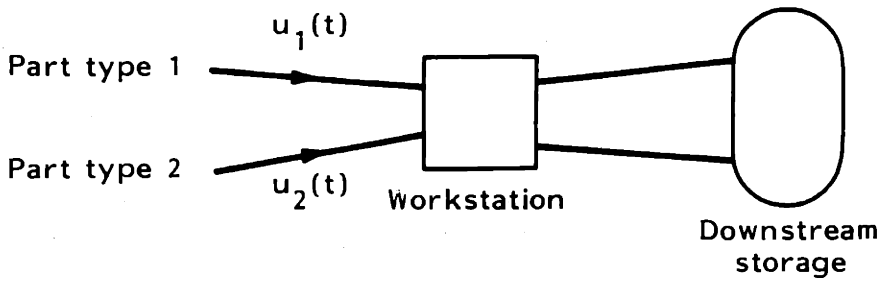

Figure 3.1 A Single Workstation System

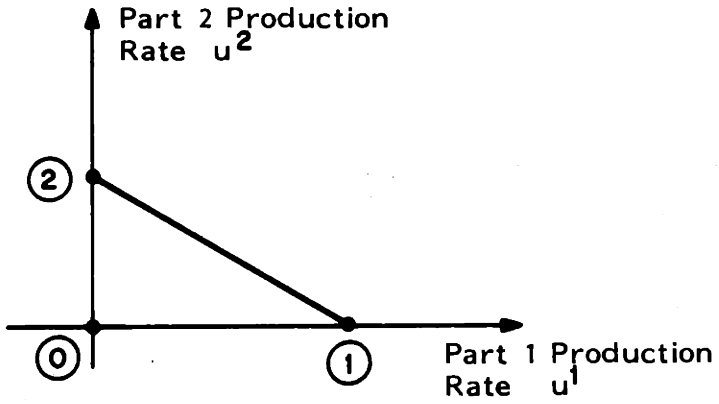

Figure 3.2 The Control Constraint set $\Omega(1)$ When the Station is Operational

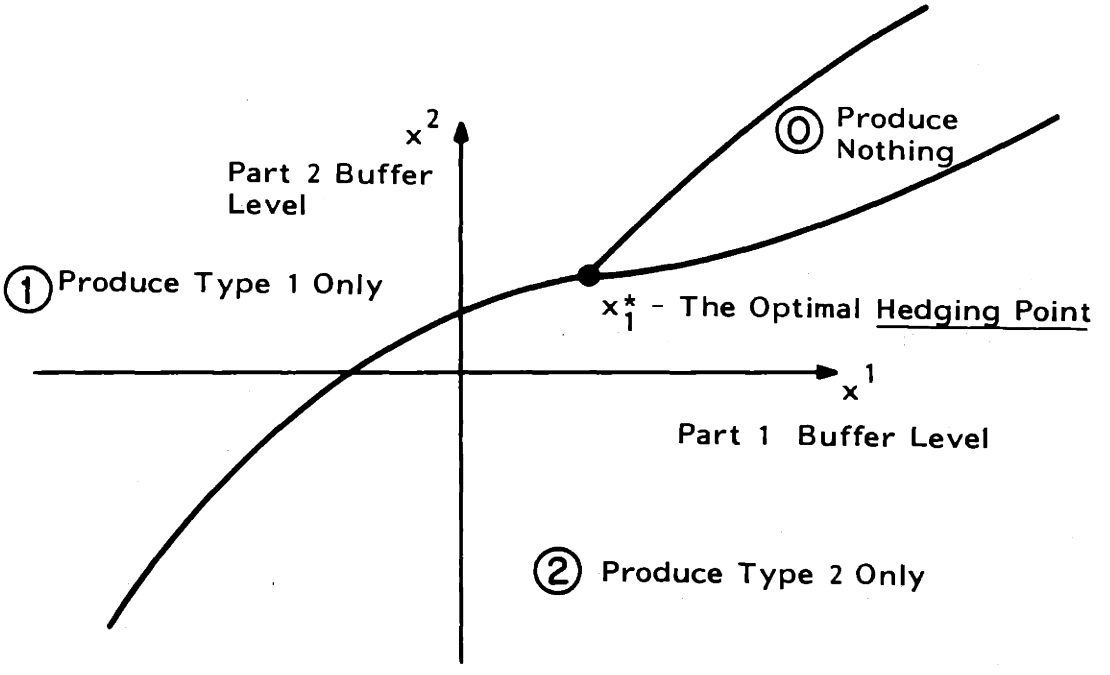

Figure 3.3 The Regions Corresponding to Extreme Points of Figure 3.2

3.4 The Continuous Time Discounted Cost Control Problem

The discounted cost index control problem is given by

$$J^*(x,i) = \min E \left\{ \int_0^\infty e^{-\beta t} g(x(t))dt \mid x(0) = x, \alpha(0) = i \right\} \quad (3.17)$$

$$\text{subject to } x(t) = u(t) - d \quad (3.18)$$

$$\text{and } u(t) \in \Omega(\alpha(t)) \quad (3.19)$$

The cost function in (3.17) is exponentially weighted. The immediate cost in terms of deviation of production from desired values is thus more important than the future cost. The discount factor β is a positive constant and the downstream demand $d(t) = d$ is constant over time.

Under appropriate conditions on the stochastic processes, it can be shown that optimal policies are time invariant. The admissible set of control policies therefore consists of functions $u: R^N \times S \to \Omega(\alpha)$, with the continuity properties described in Section 3.2. The policy thus divides R^N into a set of regions in which $u(x,i)$ is a continuous function of x. Given an admissible control law and an initial condition (x,i), the total discounted cost is given by

$$J_u(x,i) = E \left\{ \int_0^\infty e^{-\beta t} g(x(t))\, dt \mid x(0) = x, \alpha(0) = i \right\} \quad (3.20)$$

which can be written as [Kushner, 1971]

$$J_u(x,i) = E\left\{\int_0^t e^{-\beta s} g(x(s))\, ds \mid x(0) = x,\ \alpha(0) = i\right\}$$

$$+ E\left\{e^{-\beta t} J(x(t),\alpha(t)) \mid x(0) = x,\ \alpha(0) = i\right\}$$

Re-arranging and dividing through by t,

$$\frac{E\left\{J(x(t),\alpha(t)) \mid x(0) = x,\alpha(0) = i\right\} - J(x,i)}{t}$$

$$+ \frac{E\left\{(e^{-\beta t}-1)J(x(t),\alpha(t)) \mid x(0) = x,\alpha(0) = i\right\}}{t}$$

$$+ \frac{1}{t} E\left\{\int_0^t e^{-\beta s} g(x(s))ds \mid x(0) = x,\alpha(0) = i\right\} = 0 \qquad (3.21)$$

For the Markov process $z(t) = (x(t), \alpha(t))$, there corresponds an operator L (the weak infinitesimal operator) which maps the space of real bounded functions onto itself. It is defined as

$$L(f)\ (z) = \lim_{t\to 0^+} \frac{E[f(z(t)) \mid z(0) = z_o] - f(z_o)}{t}$$

with a domain consisting of functions f(z) for which the limit exists. If $J_u(x,i)$ is in the domain of L, then we can take limits in (3.21) to obtain [Kushner, 1971]

$$L(J)\ (x,i) - \beta J(x,i) + g(x) = 0 \qquad (3.22)$$

In the flow control problem, the operator L can be shown to be [Tsitsiklis, 1982].

$$L(f)\ (x,i) = \frac{\partial}{\partial x} f(x,i)[u(x,i) - d] + \sum_{j \in S} \lambda_{ij} f(x,j)$$

and thus depends on the control policy in effect.

The expression (3.22) then becomes

$$\beta J(x,i) = g(x) + \frac{\partial}{\partial x} J(x,i)[u(x,i) - d] + \sum \lambda_{ij} J(x,j) \qquad (3.23)$$

For (3.23) to hold, a necessary condition is [Kushner, 1971]

$$\lim_{t \to \infty} e^{-\beta t} E\left\{ J(x(t),\alpha(t)) \mid x(0) = x, \alpha(0) = i \right\} = 0$$

for all $x \in R^N$ and $i \in S$.

From (3.18) and (3.19), for any sample path of $\alpha(t)$ the corresponding sample trajectory of x is bounded above and below by a linear function of time. A sufficient condition for the existence of $J_u(x,i)$ for any admissible control law can consequently be stated as

$$\lim_{|y| \to \infty} e^{-\beta |y|} g(y) = 0$$

Many cost functions of interest have this property. It is then possible to have a finite discounted cost even though the buffer state becomes unbounded with probability one. For example, consider applying the control $u(x,i) = 0$ for every $x \in R^N$ and $i \in S$. In this case, for any exponentially bounded cost function $g(x)$ satisfying (3.10), a finite discounted cost exists. This is an undesirable feature because in addition to finding the optimal control law, one must also determine whether or not the buffer state remains bounded.

Under appropriate conditions, an optimal control law u^* and cost $J_{u^*}(x,i)$ satisfies [Kushner, 1971]

$$\beta J_{u^*}(x,i) = \min_{u\in\Omega(i)} \left\{ g(x) + \frac{\partial J_{u^*}(x,i)}{\partial x}[u-d] + \sum \lambda_{ij} J_{u^*}(x,j) \right\} \tag{3.24}$$

The optimal control $u^*(x,i)$ therefore divides R^N into time invariant regions in which $u^*(x,i)$ is constant at an extreme point of $\Omega(i)$. A trajectory of $x(s)$ thus moves under the action of a control which is piece-wise constant except when $x(s)$ moves along the boundary of a set in which case the control $u^*(x(s))$ is time varying. The convexity of $J_{u^*}(x,i)$ can be shown using the argument given in the finite horizon case.

3.5 The Minimum Average Cost Problem

We are interested in minimizing the average long term average value of the cost index. The performance index to be minimized is formally defined as

$$\gamma_u^i(x) = \min_{T\to\infty} E\left\{ \frac{1}{T} \int_0^T g(x(t))dt \;\middle|\; x(0) = x,\ \alpha(0) = i \right\} \tag{3.25}$$

The average cost problem has technical difficulties not encountered in the discounted cost and finite horizon cases. The limit in (3.25) must be shown to exist and be independent of initial conditions. For the limit to exist, the buffer state must be stable in the sense that it remains bounded for all time.

The characterization of optimal policies requires that a value function should exist and be well defined over the state space for all admissible policies.

Let $(x_u(t),\alpha(t))$ be the system trajectory that results when the control policy u is applied. We require that for all admissible policies, the stochastic process $(x_u(t),\alpha(t))$ is ergodic.

The demand d should be such that for any negative initial buffer state x, the trajectory $x_u^i(s;x,t)$ crosses into the positive orthant if all machines stay operational for a long enough period.

These conditions imply that admissible policies maximize production whenever production lags behind demand. There is also an implicit requirement that the demand rate is such that the system has enough spare capacity to recover from blocklogs caused by failures.

Ergodisity of the stochastic process $(x_u(t),\alpha(t))$ implies that there is a probability distribution function $P_u(x,\alpha)$ corresponding to each admissible policy. It follows that for each policy, the limit in (3.25) exists, is independent of initial conditions and is given by

$$\gamma_u = \int_{R^N x S} g(x)d\ P_u(x,\alpha) \tag{3.26}$$

The optimal policy satisfies

$$\gamma_{u*} = \inf_u \gamma_u \tag{3.27}$$

In order to derive local optimality conditions similar to (3.24), the existence of value functions $J_u(x,i)$ for each admissible policy must be demonstrated.

A sufficient condition for the existence of a value function $J_u(x,i)$ is that there be a point (x_0,i_0) in the state space that is visited by $(x_u(t),\alpha(t))$ infinitely often for all admissible policies [Tsitsiklis, 1982]. Alternatively, for a compact subset Γ of $R^N xS$ and any admissible policy u, let $X_\ell^u = (x_u(T_\ell),\alpha(T_\ell)) \in \Gamma$ be the value of the state at ℓth visit to Γ. If $\{X_\ell^u\}$ is a uniformly recurrent Markov process over Γ, then well defined value functions exist [Kushner, 1978].

The set of admissible policies in general may result in disjoint ergodic classes for the process $(x_u(t),\alpha(t))$. For example, consider the system in Figures 3.1 - 3.3 with the same feasible demand rate for the two parts. Figure 3.4 shows the regions corresponding to two admissible policies u_1 and u_2. For each policy, the hedging points $x_{u_1}^*$ and $x_{u_2}^*$ are recurrent when the machine is operational.

Once the trajectory is at the hedging point, a machine failure means that no production is possible and the trajectory follows the dashed line as the buffer stock is depleted. When the machine is repaired, policy u_1 produces type 2 parts only which drives the buffer state trajectory to the solid region boundary. Policy u_2 produces type 1 parts and thus drives the trajectory towards its boundary. Thus under each policy, the trajectory is confined to the shaded regions which are disjoint. It is therefore not possible to construct a value function with which we can compare the two policies locally. Note that the expression (3.27) still provides a global comparison for the policies.

To characterize optimal policies, the buffer state is articifially bounded above and below. Consider the situation in which the demand d^n for part n goes to zero whenever the buffer state x^n reaches $-b_n$ where

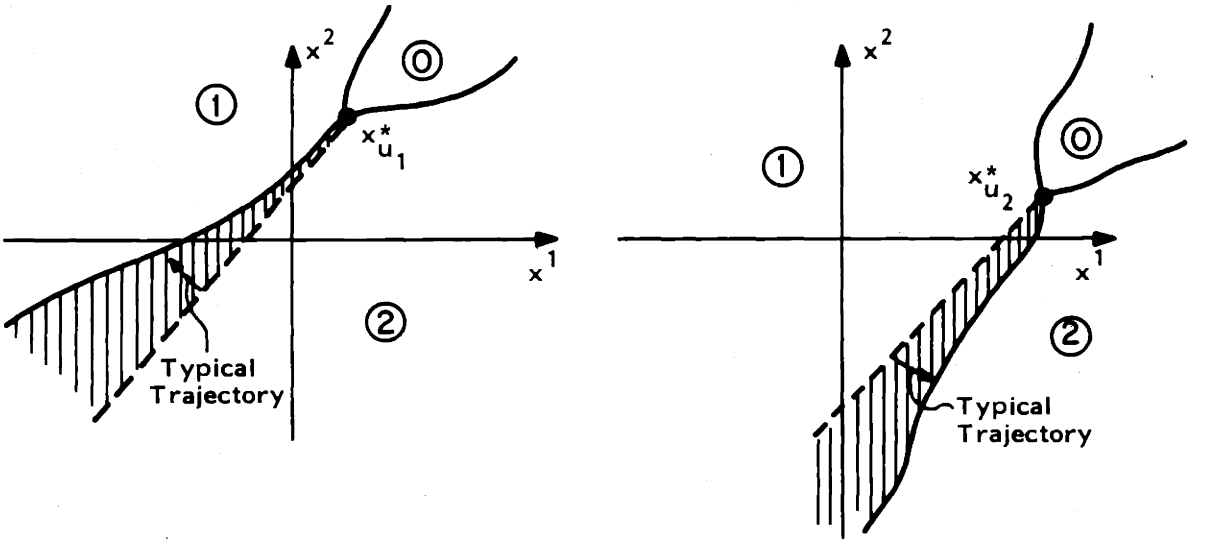

Figure 3.4 Time Invariant Regions Corresponding To Policies u_1 and u_2

b_n is some large value. Likewise, production is halted if the inventory for the part is b_n. Let i_0 be the machine state $(0,0,\ldots,0)$ with all workstations failed. For any admissible policy, if all machines are failed for a long enough period, the buffer state $x_0 = (-b_1,\ldots,b_N)$ is reached with probability 1. The point (x_0,i_0) is therefore recurrent for all admissible policies. Define the value function $J_u(x,i,t)$ as [Tsitsiklis, 1982]

$$J_u(x,i) = E\left\{\int_0^{T_1^u(x,i)} [g(x_u(t)) - \bar{\gamma}_{u^*}]dt \mid x(0) = x,\alpha(0) = i\right\} \quad (3.28)$$

where $T_1^u(x,i)$ is the first time that the system state is (x_o,i_o) given that the initial state is (x,i) and $\bar{\gamma}_{u^*}$ is the average value of the cost index corresponding to an optimal policy with the bounded state space.

An optimal control u^* and value function $J_{u^*}(x,i)$ then satisfies [Tsitsiklis, 1982]

$$\gamma_{u^*} = \min_{u(x,i)\in\Omega(i)}\left\{g(x) + \frac{\partial}{\partial x} J_{u^*}(x,i)'[u(x,i)-d] + \sum\lambda_{ij}J_{u^*}(x,j)\right\} \quad (3.29)$$

Moreover, the optimal value function is convex. The optimal policy is thus a switching policy with a connected set of regions corresponding to each vertex of the constraint set $\Omega(\alpha)$.

The bounds b_n can be made arbitrarily large. The probability that the buffer state is on the boundary under a reasonable operating policy is correspondingly small. The error introduced by bounding the state space is thus small. We might also add that any numerical scheme for computing the optimal control or evaluating the cost corresponding to a policy will have to be limited to a suitable bounded region of the state space. The approx-

imation described here thus fits into a computational framework.

3.6 Summary

The flow control model for the manufacturing system has been developed and the production control problem formulated. The production rate is constrained for each configuration of failed and operational workstations to lie in a convex polyhedral control constraint set.

Optimal policies are characterized for each machine state by a partition of the buffer state into a set of regions within which production is at an extreme point of the control constraint set. The regions are time varying in the finite horizon case.

The flow control algorithm sets piece-wise constant production rates. The routing and sequencing algorithms thus only calculate path flows and schedules when the through put is charged. The exception is when the buffer state is at a region boundary in which case the production rates may be time varying.

The computation of the optimal flow control policy is possible only for small problems because of the dimension of the state space. Practical methods for computing reasonable sub-optimal policies are needed if the hierarchical controller is to be implemented.

In the next Chapter, sub-optimal control policies which preserve the switching characteristics of the optimal control law are described.

CHAPTER IV

4.0 SUB-OPTIMAL CONTROL OF THE WORKCENTER

4.1 Introduction

The size of the state space in the flow controller makes it impractical to calculate the exact optimal production policy. Control schemes with reduced computational requirements are necessary if the method is to be implemented.

In the workcenter, any control used must lie within the control constraint set. Failure to satisfy this requirement means that material is fed into the system faster than it can be processed. The result is congestion within the system and an uncertain production rate.

In developing a sub-optimal controller, full information on the machine state must be used. This still gives rise to difficulties because of the size of the machine state space.

In this chapter, several techniques are considered which provide approximations to the optimal control policies described in Chapter III. Section 4.2 describes controllers that do not make use of buffer state information. They are included for completeness and because they are closely related to the flow optimization methods of Kimemia and Gershwin [1980] and Hildebrandt [1980].

Controllers with full state feedback are discussed in Section 4.3 and 4.4. Feedback information on the buffer state gives the controller robustness with respect to modeling errors and it is therefore useful to incorporate full state feedback in the control policy.

A policy that makes use of estimates of the cost-to-go functions $J_{u^*}(x,i,t)$ of Chapter III is discussed in Section 4.3. The calculation of the estimates involves the computation of upper and lower bounds for the cost function. These bounds are useful also in the production planning process.

A certainty equivalent control scheme is developed in Section 4.4. The stochastic machine state trajectory is replaced by a set of deterministic trajectories based on mean times to repair and to fail. The method is similar to rolling horizon techniques in inventory control problems [Blackburn and Millen, 1980]. It may also be viewed as a pruning of the machine states to leave only the most likely states in the neighborhood of the current state.

Full state feedback sub-optimal controllers described in this chapter retain the structure of the optimal policies. For each machine state i, the buffer state R^N is divided into a set of regions in which the control is at an extreme point of $\Omega(i)$. If the performance of the system is not sensitive to the location of the region boundaries, we expect that the sub-optimal policies will perform well.

4.2 Controllers that are Open Loop with Respect to the Buffer State

In this section, controllers that do not make use of buffer state information are examined. Such control schemes are appealing because once the machine state is known, the system evolves in a deterministic manner. However, they are sensitive to modeling errors and in practice, frequent operator intervention would be necessary to correct inbalances in part production.

The continuous time problem is discretized using a small discretization

interval h. In any period, the control remains constant. Given a sequence of controls $\{u_k\}$ at times $\{kh\}$, k = (0,1,2,...,) the buffer level x_k is given by

$$x_k = x_0 + \sum_{\ell=0}^{k-1} h(u_\ell - d_\ell) \tag{4.1}$$

The original production control problem can now be written as

$$\text{minimize } J(i) = \min E\left\{\sum_{k=0}^{T-1} g_k\left(x_0 + h\sum_{\ell=0}^{k-1}(u_\ell - d_\ell)\right) \mid \alpha_0 = i\right\} \tag{4.2}$$

$$\text{subject to } u_k(\alpha) \in \Omega(\alpha) \tag{4.3}$$

The minimization is with respect to variables $u_k(i) \in \Omega(i)$ which give a production rate to be used if the machine state is i during period k. The control in any period depends only on the current machine state and does not depend on the history of failures and repairs.

Provided that the expectation in (4.2) can be evaluated, this is now a deterministic mathematical programming problem which gives a control $u_k(\alpha) \in \Omega(\alpha)$ which is feasible. The only buffer state information required is the initial level x_0. Buffer state information can be updated each time there is a change of machine state. The problem (4.2) - (4.3) is then resolved with the new initial conditions. However, formidable problems still exist. The size of the machine state, the expectation in (4.2), and the constraint (4.3) yield a non-linear program with hundreds, perhaps thousands of variables.

One area where this approach can be used is in maximizing the average production rate subject to a constraint on the ratio of parts produced. This problem can be written as

$$\text{maximize} \lim_{T\to\infty} E\left\{\frac{1}{T}\sum_{k=0}^{T-1} c'u_k \,|\, \alpha_0 = i\right\} \tag{4.4}$$

$$\text{subject to} \quad E\left\{e_n'u_k\right\} = \beta_n \sum_n E\left\{e_n'u_k\right\} \quad \forall\, n \tag{4.5}$$

$$\text{and } P(\alpha_{k+1} = j \,|\, \alpha_k = i) = \lambda_{ij} \tag{4.6}$$

in which c is a vector that weights the production of the individual parts. The vector e_n in (4.5) is the nth column of the NxN identity matrix. The constraints (4.5) require that the expected production of part n during period k be a fraction β_n of total expected production in the same period. Equation (4.5) can be written as

$$e_n' \sum_{j\in S} \Pi_{ij}(k)\; u_k(j) = \beta_n \sum_n \sum_{j\in S} \Pi_{ij}(k) e_n'u_k(j) \tag{4.7}$$

where $\Pi_{ij}(k) = P(\alpha_k = j \,|\, \alpha_0 = i)$ and can be calculate independently since the failure and repair rates do not depend on the production rate.

It can be shown [Bertsekas, 1976] that there is a steady state control law $\hat{u}$ a constant J and functions f(j),(jεS) which satisfy

$$J + f(i) = \max_{u\in\Omega(i)} \; c'u + \sum_{j\in S} \lambda_{ij} f(j) \tag{4.8}$$

$$\text{subject to} \sum_{j\in S} \Pi_j e_n'u(j) = \beta_n \sum_{j\in S} \sum_n \Pi_j e_n'u(j) \tag{4.9}$$

where $\Pi_j = \lim_{k\to\infty} \Pi_{ij}(k)$ and J is the average long term production rate. The limit Π_j exists because the machine state is a finite state ergodic Markov chain. We note from (4.8) that the maximum production rate can be found by a linear program. In the absence of failures, this is identical to the deterministic processing time problem solved by Kimemia and Gershwin [1980].

Hildebrandt [1980] uses a controller that is open-loop with respect to the buffer state to minimize the time required to meet a given production target. The problem is an optimum stopping time problem and can be stated as

$$\min_{u_k \varepsilon \Omega(\alpha_k)} E\left\{ \sum_{k=0}^{\infty} g(x_k) \mid \alpha_0 = i,\ x_0 = 0 \right\} \tag{4.10}$$

$$\text{subject to } x_{k+1} = x_k + hu_k \tag{4.11}$$

the production target for part n is Q^n. Production is halted once the total production x_k is equal to or greater than $Q = (Q^1, Q^2, \ldots, Q^N)$ componentwise. The cost function is defined as

$$g(x_k) = \begin{cases} 0 & \text{if } x_k > Q \\ 1 & \text{Otherwise} \end{cases} \tag{4.12}$$

For the discretized problem, x_k and u_k take values in finite countable sets. For each part there is a machine state in which it can be produced. It can then be shown that there is a time-invariant control policy which minimizes the expected time to reach the production target.

The equivalent open loop problem can be stated as an optimization problem in the variables $u_k(j)$ and T:

$$\min T \tag{4.13}$$

$$\text{such that} \quad E\left\{\sum_{k=0}^{T-1} u_k(\alpha_k) \mid \alpha_0 = i\right\} \geq Q \tag{4.14}$$

$$\text{and } u_k(j) \in \Omega(j) \tag{4.15}$$

equation (4.14) can be written as

$$\sum_{k=0}^{T-1} \sum_{j \in S} \Pi_{ij}(k) u_k(j) \geq Q \tag{4.16}$$

It is no longer guaranteed that this problem will yield a time invariant solution. However, we can simplify the problem by seeking a stationary policy u and replacing $\Pi_{ij}(k)$ by the expected proportion of time $\tau_{ij}(T)$ the machine state is j in the interval [0,T] given that the initial state is i. The problem can then be written as :

$$\min \quad T \tag{4.17}$$

$$\text{subject to} \quad \sum_{j \in S} \tau_{ij}(T) u(j) \geq Q \tag{4.18}$$

$$u(j) \in \Omega \ (j) \tag{4.19}$$

This is the problem solved by Hildebrandt [1980]. In practice, the production levels would be reviewed periodically or after a machine state change and the problem resolved for the remaining production.

Policies that are open loop with respect to the buffer state are sensitive to modelling errors particularly in the buffer state dynamics. In practice, there are sources of uncertainties other than failures and repairs which influence the buffer state trajectory. Unless they are explicitly taken into account, the performance of an open-loop controller is likely to be adversely affected. Since the policy does not depend on the current buffer state, the open loop controller does not respond to buffer level fluctuations. Large downstream buffers are therefore necessary in order to prevent the starvation of downstream manufacturing processes.

4.3 An Estimate Based (EB) Control Scheme

4.3.1 The Approach

The optimal policy in the flow control problem is determined from the optimal value function $J_{u^*}(x,i,t)$ by the linear program (3.16). The optimal policy is a feedback law which, for every value of α divides the buffer state space into a set of regions within which the control is constant at an extreme point of the control constraint set $\Omega(\alpha)$. Strict convexity of $J_{u^*}(x,i,t)$ means that the boundary separating two regions is connected and lies in a lower dimension space.

The optimal value function $J_{u^*}(x,i,t)$ is the expected cost if the system starts with a buffer level x, machine state i and operates optimally until the final time T. For a fixed buffer level, $J_{u^*}(x,i,t)$

is large for machine states i with low productive capacity. The backward iteration of dynamic programming means that all possible sequences of failures and repairs are taken into account in evaluating the cost-to-go function.

The gradient $\frac{\partial}{\partial x} J_{u^*}(x,i,t)$ can be regarded as a weighting on part production for the optimal control law. The part types that are most vulnerable to machine failures have the most weight and consequently when there is a backlog, the optimal policy is more inclined to produce them, than the less vulnerable parts. The calculation of $J_{u^*}(x,i,t)$ also takes into account the relative cost of backlogs and inventory storage determined by the cost functions $g_n(x^n)$ for part n. Thus a part that has a high value index and is at the same time sensitive to machine failures would have a correspondingly high value component in the cost-to-go function.

The hedging point $x_i^*(t)$, which is the minimum of $J_{u^*}(x,i,t)$ with respect to x, is the optimal downstream buffer levels with which to hedge against future failures. When the demand is close to the capacity of the system, the hedging points are at high buffer levels because failures quickly result in deficits and because recovery from a deficit is slow.

Given convex functions $\Psi(x,i,t)$ which are estimates of the optimal cost-to-go function, consider the control policy $\hat{u}$ determined by

$$\hat{u}(x,i,t) = \underset{u\in\Omega(i)}{\operatorname{argmin}} \frac{\partial}{\partial x} \Psi(x,i,t)'u \qquad (4.20)$$

The sub-optimal policy $\hat{u}$ like the optimal policy divides the buffer state into a set of regions in which it takes values at the extreme points of $\Omega(i)$. The sub-optimal trajectory, $x^i_{\hat{u}}(s;x,t)$ evolves under the action of piecewise constant controls except at boundaries of the regions where the control may be time varying.

The estimates $\Psi(x,i,t)$ of the cost-to-go function should exhibit the properties of $J_{u^*}(x,i,t)$ described above. The values of $\Psi(x,i,t)$ should be largest for machine states with the smallest production capacity. The relative maginitude of the components of the gradient $\frac{\partial}{\partial x}\Psi(x,i,t)$ should reflect both the relative value of the parts and their vulnerability to machine failures. The minimum of $\Psi(x,i,t)$, which determine the hedging points for the sub-optimal policy, should be of a magnitude comparable to the optimal hedging buffer levels. If the estimates satisfy these criteria, we expect the sub-optimal policy to perform well and to meet the demand requirements when they are close to system capacity. Provided that the optimal value of the cost index is not very sensitive to the location of the optimal region boundaries, the cost $J_{\hat{u}}(x,i,t)$ corresponding to the control policy $\hat{u}$ should be close to the optimal cost $J_{u^*}(x,i,t)$.

4.3.2 Calculation of Estimates $\Psi^i(x,t)$

The control constraint set $\Omega(i)$ is the intersection of a set of half spaces each containing the origin and the positive orthant. Define $\bar{H}(i)$ and $\underline{H}(i)$ to be sets such that

$$\bar{H}(i) = \left\{ u \in R^N \mid 0 \leq u^n \leq \bar{L}_{in} \right\} \qquad n = 1,2,\ldots,N \qquad (4.21)$$

$$\underline{H}(i) = \left\{ u \in R^N \mid 0 \leq u^n \leq \underline{L}_{in} \right\} \qquad n = 1,2,\ldots,N \qquad (4.22)$$

and

$$\underline{H}(i) \subseteq \Omega(i) \subseteq \bar{H}(i) \qquad (4.23)$$

$\underline{H}(i)$ and $\bar{H}(i)$ are hypercubes, the former contained in $\Omega(i)$, and the latter containing the control constraint set. For example, Figure 4.1 shows the hypercubes for the control constraint set of Figure 3.2 when the machine is operational.

Consider the stochastic control problem (3.8) with the sets $\Omega(i)$ replaced in turn by $\underline{H}(i)$ and $\bar{H}(i)$. We obtain,

$$\bar{\Psi}(x,i,t) = \min_{u(t)\in\underline{H}(\alpha(t))} E\left\{ \int_t^T g(x(s))ds \mid \alpha(t) = i \right\} \qquad (4.24)$$

and

$$\underline{\Psi}(x,i,t) = \min_{u(t)\in\bar{H}(\alpha(t))} E\left\{ \int_t^T g(x(s))ds \mid \alpha(t) = i \right\} \qquad (4.25)$$

both subject to (3.1) and (3.4).

From (4.23), the following holds

$$\underline{\Psi}(x,i,t) \leq J_{u^*}(x,i,t) \leq \bar{\Psi}(x,i,t) \qquad (4.26)$$

Thus $\underline{\Psi}(x,i,t)$ and $\bar{\Psi}(x,i,t)$ are lower and upper bounds on the optimal cost-to-go $J_{u^*}(x,i,t)$. An estimate of J_{u^*} can be obtained by taking a convex combination of the lower and upper bounds.

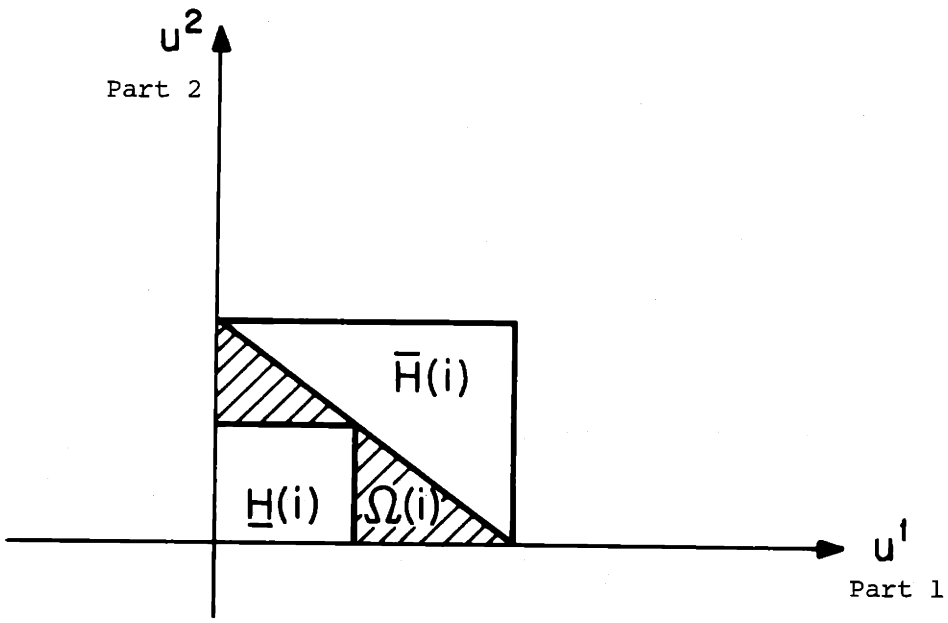

Figure 4.1 Hypercubes Corresponding to the Control Constraint set of Figure 3.1

$$\Psi(x,i,t) = (1 - \eta)\ \bar{\Psi}(x,i,t) + \eta\ \underline{\Psi}(x,i,t) \qquad (4.27)$$

where η is a scalar satisfying $0 \leq \eta \leq 1$. From (4.26), it can be deduced that

$$\left|\Psi(x,i,t) - J_{u^*}(x,t)\right| \leq \left|\bar{\Psi}(x,i,t) - \underline{\Psi}(x,i,t)\right| = \Delta_i(x,t) \qquad (4.28)$$

When the difference between lower and upper bounds is small, the estimate $\Psi(x,i,t)$ is good and we can expect the sub-optimal control to be close to the optimal policy.

The choice of η will depend on experiment and experience. Giving more weight to the upper bound $\bar{\Psi}(x,i,t)$ results in a policy that is more cautious in the sense that it has larger hedging points than one that puts greater weight on the lower bound. In simulation tests presented in Chapter VI, we have found that choosing $\eta = 0$ that is, using the upper bound as the estimate results in good performance. This is because the more cautious sub-optimal policy provides an added margin of safety against unmodelled disturbances which adversely affect production.

The hypercubes $\underline{H}(i)$ and $\bar{H}(i)$ approximate the control constraint set $\Omega(i)$. Thus if the capacity for the production of part n is small in machine state i, the corresponding limits $\bar{L}_{in}$ and $\underline{L}_{in}$ of the hypercube are small. Likewise, if the capacity is large, so are the limits $\underline{L}_{in}$ and $\bar{L}_{in}$. The calculation of the upper and lower bounds thus takes into account the relative productive capacity in all machine states,

demand rates and the value of the parts as given by the cost function $c_n(x^n)$. We expect therefore that the cost estimates satisfy the criteria of Section 4.3.1 necessary for good performance by the estimate based controller.

Using the notation of Section 4.2, (4.24) and (4.25) are discretized to yield the problem

$$\text{minimize} \quad E\left\{\sum_{k=0}^{T-1} g(x_{k+1}) + \frac{1}{2} g(x_T)\right\} \tag{4.29}$$

$$\text{subject to } x_{k+1} = x_k + h(u_k - d_k) \tag{4.30}$$

$$\text{and} \qquad u_k \in \begin{cases} \underline{H}(\alpha_k) & \text{for evaluating the upper bound} \\ \bar{H}(\alpha_k) & \text{for evaluating the lower bound} \end{cases} \tag{4.31}$$

The objective function (4.29) is obtained by taking the approximation

$$\int_t^{t+h} g(x(s;x,t))\, ds = \frac{h}{2}\left[g(x) + g(x(t+h;x,t))\right] \tag{4.32}$$

The dynamic problem (4.29) - (4.31) is solved by the dynamic program

$$\Psi_T(x,i) = \frac{1}{2} g(x_T) \quad \forall\, x \in R^N \text{ and } i \in S \tag{4.33}$$

$$\Psi_k(x,i) = \min_{u_k}\left[g(x_{k+1}) + \sum \lambda_{ij} \Psi_{k+1}(x_{k+1}, j)\right] \tag{4.34}$$

subject to (4.21) for the lower bound calculation and (4.22) for the upper bound. In (4.34), λ_{ij} is the probability of a transition from

machine state i to j during the kth period. The solution to the dynamic programming problem are estimates $\Psi_k(x,i)$ at discrete times $(0,h,2h,\ldots,hT)$.

The cost function $g(x)$ is given by (3.9) and is therefore separable. The constraints (4.21) and (4.22) affect each part separately. The dynamic program (4.29) - (4.31) is therefore decoupled and can be solved individually for each part.

It is straight forward to show that solutions of (4.34) take values at the extreme points of the hypercubes. The exception are on the surfaces separating regions of R^N with two different extreme point solutions. On these surfaces, the solution may be in the interior of the hypercube. For small discretization steps h, we can simplify the solution of the dynamic program by considering only extreme values of u_k^n i.e., 0 or $\underline{L}_{in}$ and $\bar{L}_{in}$.

4.3.3 The Infinite Horizon Case

In infinite horizon problems, we seek time invariant estimates $\Psi(x,i)$ to the optimal value functions $J_{u^*}(x,i)$ of Sections 3.4 and 3.5. In both the discounted and average cost problems, the successive approximations algorithm can be employed [Bertsekas, 1976].

Consider the discretized problem (4.29) - (4.31) with (4.29) replaced by the discounted cost

$$E\left\{\sum_{k=1}^{\infty} \beta^k\, g(x_k)\right\} \tag{4.35}$$

For computational purposes, we restrict the buffer state x_k to a bounded domain D in R^N. For the function $\Psi(x,\alpha)$: D x S $\rightarrow$ R, define the map-

pings

$$T(\Psi)(x,i) = \min_{u} E_{\alpha_{k+1}} \left\{ g(x+h(u-d)) + \beta\Psi(x+h(u-d),\alpha_{k+1}) \,|\, \alpha_k = i \right\} \qquad (4.36)$$

subject to (4.39)

and

$$T_u(\Psi)(x,i) = E_{\alpha_{k+1}} \left\{ g(x+h(u-d)) + \beta\Psi(x+h(u-d),\alpha_{k+1}) \,|\, \alpha_k = i \right\} \qquad (4.37)$$

The following results hold [Bertsekas, 1976].

(i) There exists functions $\Psi^*(x,i)$ and $u^*(x,i)$ such that

$$T_{u^*}(\Psi^*)(x,i) = T(\Psi^*)(x,i) = \Psi^*(x,i) \qquad (4.38)$$

(ii) For any bounded function $\Psi(x,i)$,

$$\lim_{k\to\infty} T^k(\Psi)(x,i) = \Psi^*(x,i) \qquad (4.39)$$

Here, u^* is a policy that minimizes (4.35) and $\Psi^*(x,i)$ is an upper bound to the cost function if the constraint set $\underline{H}(i)$ is used or a lower bound if $\bar{H}(i)$ is used. The estimate $\Psi(x,\alpha)$ is obtained by taking a weighted combination of the upper and lower bounds. There are also bounds at each iteration on $T^k(\Psi)(x,i)$.

Define

$$\bar{c}_k = \frac{\beta}{1-\beta} \max_{\substack{x\in D \\ i\in S}} [T^k(\Psi)(x,i) - T^{k-1}(\Psi)(x,i)] \qquad (4.40)$$

$$\underline{c}_k = \frac{\beta}{1-\beta} \min_{\substack{x\in D \\ i\in S}} [T^k(\Psi)(x,i) - T^{k-1}(\Psi)(x,i)] \tag{4.41}$$

It follows that [Bertsekas, 1976]

$$T^k(\Psi)(x,i) + \underline{c}_k \leq \Psi^*(x,i) \leq T^k(\Psi)(x,i) + \bar{c}_k \tag{4.42}$$

At each iteration, $\underline{c}_k$ and $\bar{c}_k$ provide lower and upper bounds, respectively, on $\Psi^*(x,\alpha)$. Consequently, at each step in the calculation of $\Psi^*(x,\alpha)$, any $u \in H(\alpha)$ such that

$$T_u T^k(\Psi)(x,i) \geq T^k(\Psi)(x,i) + \bar{c}_k - \beta \underline{c}_k \tag{4.43}$$

can be eliminated from further consideration because it can never be in the solution to the dynamic programming problem. The algorithm then terminates in a finite number of steps because initially for each point x in the buffer state, we consider a finite number of controls, the corners of H(i). At each step, (4.43) is used to eliminate some of the control values from further consideration. Eventually therefore, only those values that minimize (4.35) are left. The algorithm then terminates. Using the bounds $\underline{c}_k$ and $\bar{c}_k$, the calculation may also be terminated once $|\bar{c}_k - \underline{c}_k|$ becomes less than a preset tolerance.

The estimates for the average cost problem can be found by a modified form of the successive approximation algorithm. The discretized problem (4.29) - (4.31) has the objective (4.29) replaced by

$$\gamma = \min_{u_k} \lim_{T\to\infty} \frac{1}{T} E_{\alpha} \sum_{k=1}^{T} g(x_k) \tag{4.44}$$

Let $z \in D$ be a fixed buffer level and i_0 a fixed machine state. The estimate γ is calculated as follows

$$F_{k+1}(x,i) = \min_{u} E_{\alpha_{k+1}} \left[g(x + h(u-d)) + f_k(x + h(u-d), \alpha_{k+1}) \middle| \alpha_k = i \right] \tag{4.45}$$

subject to (4.41)

$$f_{k+1}(x,i) = F_{k+1}(x,i) - F_{k+1}(z,i_0) \qquad \forall\, i \in S$$

where $f_0(x,\alpha)$ is an arbitrary bounded function. We have [Bertsekas, 1976]

$$\gamma = \lim_{k\to\infty} F_k(z,i_0) \tag{4.46}$$

and

$$f(x,i) = \lim_{k\to\infty} f_k(x,i) \qquad \forall\, i \in S \tag{4.47}$$

where $f(x,i)$ is either an upper or lower bound to the optimal value function depending on the constraint set used and satisfies

$$\gamma + f(x,i) = \min_{u} \left(g(x+h(u-d)) + h \sum_{j\in S} \lambda_{ij} f(x+h(u-d),j) \right) \tag{4.48}$$

Bounds similar to (4.42) exist and can be evaluated along with $F_k(x,i)$ and $f_k(x,i)$.

4.3.4 Implementation of the Estimate Based Controller

There are two steps in the implementation of the EB controller. Off-line, upper and lower bounds to the optimal cost-to-go function $J_{u^*}(x,i,t)$ are computed by the dynamic programming algorithm (4.29)-(4.31). The estimates $\Psi(x,i,t)$ are calculated from (4.27) and stored. On line, whenever the system enters machine state i, the control vector u(t) is determined by the linear program (4.20) using the stored values of $\Psi(x,i,t)$.

For a production control problem with horizon T, N parts, M workstations and a discretization step length h, the off-line problem (4.29)-(4.31) involves solving 2N sub-problem, one for each part type, to obtain upper and lower bounds. Each sub-problem involves $2^M T/h$ solutions to the dynamic programming iteration (4.34) and the storage of $2^M T/h$ functions $\Psi_k(x,i)$. The computational and storage cost thus grows exponentially with the number of workstations and linearly with the number of parts. However, the estimates can be stored in peripheral storage devices. We have also found in simulation tests that fitting appropriate smooth convex functions to the numerically calculated cost estimates retains the good performance of the EB controller while reducing considerably the storage cost. The off-line computational cost can be reduced by pruning the machine state space to exclude states with low probabilities. With the failure and repair rates typically found in manufacturing systems, a small number of states account for over 95% of the probability. A large number of unlikely machine states can therefore be eliminated without substantially altering the regions and the hedging points corresponding to the sub-optimal control.

The on-line computation consists of the linear program (4.20) and has N variables and M constraints. Typically, N is between 5 and 10, and M between 10 and 20. The program can thus be easily solved on a small mini-computer.

4.4 The Certainty Equivalence (CE) Controller

A sub-optimal control scheme for the discretized problem (4.29) - (4.30) subject to $u_k \in \Omega(\alpha_k)$ can be realized by solving a set of deterministic optimization problems based on a set of machine state trajectories in real time.

Let the system enter machine state i at time t with a buffer level x. Define $S_N(i)$ to be the set of neighboring machine states. That is, $S_N(i)$ is the set of machine states that the system can reach directly from state i due to the failure or repair of a single machine. Let T_j be the expected length of time that the system resides in machine state j. At each period k of the discretized problem, with probability λ_{ij}, a transition to state $j \in S_N(i)$ takes place. For the purposes of constructing the certainty equivalent (CE) controller, it is assumed that the system actually stays in state j for the expected number of periods T_j.

Let x_k be the buffer state at beginning of the kth period and u_{jk} the control that is applied if the system enters machine state j during the kth period. Given that the system stays in state j for T_j periods, the buffer state at time $k + T_j$ is given by

$$x_{k+T_j} = x_k + T_k u_{jk} - d_{kj} \tag{4.49}$$

where d_{kj} is the total demand in the interval $[k,k+T_j]$ and is given by

$$d_{kj} = \sum_{\ell=1}^{T_j} d_{k+\ell} \tag{4.50}$$

The cost index associated with buffer state x_k at period k is taken to be

$$g(x_k) + \sum_{j\epsilon S_N(i)} \lambda_{ij} g(x_{k+T_j}) \tag{4.51}$$

Thus the cost function $g(x_k)$ is modified by the possibility of a failure or repair raking the system to state j and remaining there for T_j periods.

The probability that the system remains in state i for k periods is given by λ_{ii}^k. The probability of first entering state j during the (k+1)st period is $\lambda_{ii}^k \lambda_{ij}$. At time t, if the buffer state is x(t) and the machine state is $\alpha(t) = i$, a sequence of controls $\{u_k \epsilon \Omega(i)\}$ $k = 0,1,\ldots,T_i-1$ can be obtained by solving the deterministic problem

$$\text{minimize} \quad \sum_{k=0}^{T_i-1} \lambda_{ii}^k \left[g(x_k) + \sum_{j\epsilon S_N(i)} \lambda_{ij} g(x_{k+T_j}) \right] + g_{T_i}(x_{T_i}) \tag{4.52}$$

$$\text{subject to} \quad x_{k+1} = x_k + h(u_k - d_k) \tag{4.53}$$

$$u_k \epsilon \Omega(i) \tag{4.54}$$

$$u_{kj} \epsilon \Omega(j) \tag{4.55}$$

$$x_0 = x(t) \tag{4.56}$$

The sequence $\{u_k\}$ is applied until either the expected number of periods T_i in state i is reached or a change of machine state takes place. In either case, the calculation is repeated to obtain a new sequence of controls.

The cost function (4.52) hedges against the possibility of failures by weighting the possible loss of production due to a transition to state j by the probability of that transition. The terminal cost $g_{T_i}(x)$ compensates for the short horizon and for the fact that only transitions to neighboring states are included in the formulation. Using $g_{T_i}(x)$, we can penalize the controller for the final buffer state x_{T_i}, thus ensuring that subsequent control sequences start from more favorable buffer state values.

The number of variables in the problem is given by

$$2NT_i(1+2M) \tag{4.58}$$

where N is the number of parts and M the number of machines. The number of constraints is

$$NT_i + M(1+2MT_i) \tag{4.59}$$

The size of the optimization problem therefore grows linearly with the number of parts and machines.

In practice, the size of the problem can be quite large. For example, if there are four parts, ten machines and the expected time in the current state is divided into five periods (i.e., N=4, M=10 and T_i=5), there are 840 variables and 1030 constraints. A problem of this size may impose a considerable computational burden on the workcenter control computer, par-

ticularly as it may have to solve the optimization problem as often as every 10-15 minutes. Additional simplification is necessary before the certainty equivalent controller can be applied in situations where computer facilities for on-line control are limited.

Optimization problem (4.52) - (4.56) considers the possible buffer state trajectory $\{x_k\}k=1,2,\ldots,T_i-1$ in the current machine state and the final state x_{T_i} in chosing the control sequence $\{u_k\}k=0,\ldots,T_i-1$. The size of the optimization problem can be considerably reduced if we only consider the immediate buffer level x_{k+1} due to the control u_k applied at period k and the possible final buffer levels x_{k+T_j} that can be reached from x_k. We motivate the approach by looking at the dynamic programming formulation of (4.52) - (4.56).

Define

$$J_k(x) = \min \sum_{\ell=k}^{T_i} \lambda_{ii}^{\ell-k} \left(g(x_\ell) + \sum_{j\in S_N(i)} \lambda_{ij} g(x_{\ell+T_j})\right) \qquad (4.59)$$

subject to (4.53 - (4.56)

and $\quad x_k = x \qquad (4.60)$

Application of the dynamic programming recursion to (4.59) yields

$$J_k(x_k) = \min \left[g(x_k) + \sum_j \lambda_{ij} g(x_{k+T_k}) + \lambda_{ii} J_{k+1}(x_{k+1})\right] \qquad (4.61)$$

subject to (4.53) - (4.56)

$$\text{with} \quad J_{T_i}(x_{T_i}) = g_{T_i}(x_{T_i}) \tag{4.62}$$

We now assume further that the system stays in machine state i until time T_i if no transitions occur in the (k+1)st period. Since we consider the value of the cost index at the final period T_i, we can replace $J_{k+1}(x_{k+1})$ by

$$\min_{x_{T_i}} g_{T_i}(x_{T_i}) \tag{4.63}$$

Thus if at period k the machine and buffer states are α_k=i and x_k=x respectively, we solve one step of the dynamic program

$$J_k(x)=\min\ g(x_{k+1}) + g_{T_i}(x_{k+1} + T_i u_{ik} - d_{ik}) + \sum_{j\in S_N(i)} g_{T_j}(x_{k+1} + T_j u_{jk} - d_{jk}) \tag{4.64}$$

$$\text{subject to} \quad x_{k+1} = x_k + u_k - d_k \tag{4.65}$$

The control u_k is applied for one period and the problem is resolved for the new buffer state x_{k+1}. The simplified procedure penalizes only the immediate buffer state x_{k+1} and the possible final state x_{T_i} and x_{T_j}. The original CE controller penalizes the entire trajectory $\{x_k\}$ k = 0,...,T_i-1 while in machine state i.

The optimization problem (4.64) - (4.65) has 2N(M+1) variables and

$2M(M+1)+N$ constraints. However, it is solved more often than the original CE controller.

There are other choices of possible state trajectories that can be considered in constructing CE control policies. Some experimentation is necessary in order to find machine state trajectories that give the best hedging behavior for the CE policy.

Certainty equivalence controllers do not take into account the possibility of a succession of failures since only a single machine state transition is considered. As a result, the policy might not maintain a sufficient number of parts in the buffers as a hedge against machine failures. By replacing $g_{T_i}(x)$ with $g_{T_i}(x-y)$, a buffer stock y is imposed on the controller. The performance of the controller may be improved, but the choice of y will depend on experience and experimentation.

4.5 Summary

In this Chapter, we have developed practical schemes for computing flow control policies in an automated manufacturing system. All three methods solve in real time an optimization problem whose solution yields a control trajectory.

The deterministic nature of the buffer state for a fixed machine state is utilized in the schemes of Section 4.3. The control policy is a function only of the machine state and is determined off-line and stored. The fact that this method is open loop with respect to the buffer state means that the control policy is sensitive to modelling errors. In practice, frequent updates requiring the solution of (4.2) - (4.3) would be necessary

making the method less attractive. Related methods have been proposed for flow optimization of flexible manufacturing systems in the absence of failures [Kimemia and Gershwin, 1980] and for minimizing the time to produce a given quantity of material in a system with unreliable workstations [Hildebrandt, 1980].

The Estimate Based control scheme is described in Section 4.3. An off-line calculation provides upper and lower bounds to the optimal cost-to-go functions which are used on-line to give full state feedback control of the workcenter. EB controllers take into account all possible sequences of failures and repairs. This results in a hedging behavior which is comparable to that of the optimal control policy. The computational and storage requirements grow exponentially with the number of machines which limits the size of system for which this method can be applied. The computational difficulty can be reduced by eliminating unlikely machine states. This is demonstrated in Chapter VI.

Section 4.4 describes certainty equivalence (CE) controllers. The control is found on-line by solving an optimization problem based on a set of deterministic machine state trajectories. CE controllers have the advantage of not requiring costly off-line computation and yet retaining full state feedback. Since only immediate failures and repairs are included in the calculations, CE controllers might not hedge adequately against failures unless additional precautions are taken. The on-line computation may also be frequent and costly to perform.

CHAPTER V

5.0 PRODUCTION PLANNING FOR AN AUTOMATED MANUFACTURING SYSTEM

5.1 Introduction

The hierarchical production control algorithm is designed to regulate the flow rate of parts through a manufacturing system so that production goals are met. It is important for the performance of the system that the production requirements should be feasible. That is, the requirement should be within the capacity of the system. At the same time, in order to maximize utilization, the system should be operated as close as possible to capacity. The managers of the workcenter should therefore have planning tools that ensure that demand is not only feasible but also leads to maximum utilization.

The flexibility of a computer controlled manufacturing system enables it to process simultaneously different members of a part family. The capacity of the system is defined in Section 3.2 by production constraint sets $\Omega(i)$ which fully describe all the possible combinations of part flow rates that the system can produce. Knowledge of these sets is important not only to the managers of the workcenter but also for production planners who make decisions concerning the production process weeks or months in advance.

In Section 5.2 a simple procedure is derived for ascertaining whether or not a given demand is feasible. We conjecture that feasibility of demand is sufficient to ensure the existence of the minimum average cost control problem of Section 3.5. The production planning problem for workcenter managers is defined and an example outlined in Section 5.3.

The procedure of Section 5.2 involves verifying whether or not a given demand requirement lies within the capacity set of the workcenter. The calculation of the capacity set depends on the values of system parameters which are not known accurately. Furthermore, there may be additional unmodelled factors which affect the capacity of the system. In our simulation study reported in Chapter VI for example, the simple interval loading scheme implemented in the sequence control level of the hierarchical controller is only able to achieve approximately 90% utilization of available time at bottleneck workstations. The result is a failure to satisfy demand requirements set using apriori estimates of system capacity which assume 100% utilization of bottleneck stations. However, the resulting lower than expected production rate is evident early in the simulation run as it is shown in the buffer state plot of Figure 6.11. It is important therefore that production planners at the workcenter continuously monitor the performance of the system. Using predictive and estimation algorithms discussed in Section 2.5, they should revise their production plans and control policies if it becomes evident that production targets cannot be met. For this reason, Section 5.3 recommends dynamic planning techniques which react to actual workcenter output.

5.2 The Equivalent Production Capacity Set

5.2.1 The Infinite Horizon Case

We consider first, the infinite horizon production control problem described in Section 3.4 and 3.5. For a given admissible control policy u, initial conditions $x(0) = x$ and $\alpha(t)=i$, the system state $(x_u(t),\alpha(t))$ is an ergodic random process. Furthermore, the set of admissible control

policies described in Section 3.5, admits a single ergodic class in the state space R^N x S. For each policy u, therefore, there is a corresponding probability distribution $P_u(x,\alpha)$ defined over the state space R^N x S which is independent of initial conditions.

The mean production rate in machine state i is defined as

$$\bar{u}(i) = \int_{R^N} u(x,i)\ dP_u(x,i) \tag{5.1}$$

The convexity of the control constraint sets $\Omega(i)$ implies that the mean production $\bar{u}(i)$ is within the set $\Omega(i)$ for all machine states. The overall average production rate is given by

$$\bar{u} = \underset{\alpha}{E}(\bar{u}(\alpha)) = \sum_{i\varepsilon S} \Pi_i \bar{u}(i) \tag{5.2}$$

where Π_i is the steady state probability that the system is in machine state i. Given a demand on the system, we say that it is feasible if there is an admissible policy for which

$$d = \bar{u} \tag{5.3}$$

The equivalent production capacity set Φ is defined as the set of all d such that d is feasible. Let w_i be a vector of workstation flow rates of type n parts undergoing operation k at station m w_{nm}^k corresponding to the production rate u_i. We can state the relationship (3.5) and (3.6) in matrix notation as

$$u_i = Bw_i \qquad \forall\, i \in S \tag{5.4}$$

and

$$Aw_i \leq b(i) \tag{5.5}$$

where B is a matrix with elements 0 or 1, A is the matrix of operation times for type n parts undergoing operation k at station m τ^k_{nm} and b(i) denotes the right hand side of (3.6) for machine state i. Combining (5.3) - (5.5), it follows that d is in Φ if a set of operation splits w_i can be found such that

$$d = B \sum_{i \in S} \Pi_i w_i \tag{5.6}$$

and

$$A \sum_{i \in S} \Pi_i w_i \leq \sum_{i \in S} \Pi_i b(i) \tag{5.7}$$

The feasibility of a demand requirement d can be verified by finding average workstation flow rates $\bar{w}$ that satisfy

$$d = B\bar{w} \tag{5.8}$$

$$A\bar{w} \leq \bar{b} \tag{5.9}$$

where $\bar{b} = \sum_{i \in S} \Pi_i b(i)$. From the definition of b(i), $\bar{b}$ is the expected machine state. We can interpret conditions (5.8) and (5.9) as replacing the workcenter with an equivalent manufacturing system defined by

the expected machine state. This is equivalent to replacing each machine by a reliable machine with a capacity reduced by p/(r+p) where 1/p is the mean time between failures and 1/r the mean time to repair.

Given any $u_i \varepsilon \Omega(i)$, $i \varepsilon S$, we can always find an admissible policy u such that

$$\bar{u}(i) = u_i \tag{5.10}$$

one obvious choice is $u(x,i) = u_i \ \forall \ x \varepsilon R^N$ and $i \varepsilon S$. In fact, we could restrict our attention to policies that take values at extreme points of $\Omega(i)$ since each set is covered by a convex combination of its extreme points.

This section has shown that a constant demand rate d in the infinite horizon problems of Sections 3.4 and 3.5 is feasible or in other words can be satisfied if and only if it belongs to the equivalent capacity set Φ. To verify whether or not a demand is feasible, involves solving the set of linear inequalities (5.8) - (5.9) which can be done fairly easily.

5.2.2 The Finite Horizon Case

The definition of the feasible production set Φ can be extended to the finite horizon flow control problem. The set of feasible production requirements then depends on the initial buffer state x, machine state i and the interval (0,T) over which the demand is calculated.

The concept of feasibility in the finite horizon case differs from the infinite horizon control problem. For the finite horizon problem, the equivalent production capacity is the set of expected production rates that can be achieved over the interval (0,T) when failures and repairs are taken into account. It is possible that the mean production rates for samples of the machine state trajectory $\alpha(t)$, $t\varepsilon(0,T)$ are outside the equivalent production capacity set. In the infinite horizon case for admissible control policies, the limit of the expected production rate in the interval (0,T) lies in Φ as T becomes infinitely large for all initial conditions. The distinction between expected and sample production rates therefore, does not arise.

Let d(t) be a specified deterministic time varying demand rate with the average d_T given by

$$d_T = \frac{1}{T}\int_0^T d(t)\,dt \tag{5.11}$$

For an admissible policy $u \varepsilon U_A$, define the mean production rate in machine state j given the initial conditions $x(0) = x$, $\alpha(0) = i$ as

$$\bar{u}_{ij}(x,T) = \frac{E\left\{\int_0^T u(x_u(t),\alpha(t))\delta(\alpha(t),j)dt \mid x(0)=x,\alpha(0)=i\right\}}{E\left\{\int_0^T \delta(\alpha(t),j)dt \mid x(0)=x,\alpha(0)=i\right\}} \tag{5.12}$$

where $\delta(t,j)$ is an indicator function for machine state j given by

$$\delta(\alpha(t),j) = \begin{cases} 1 & \text{if } \alpha(t) = j \\ 0 & \text{Otherwise} \end{cases}$$

Thus $\bar{u}_{ij}(x,T)$ is the expected amount of material produced while the system is in machine state j divided by the expected time in the machine state.

An average requirement d_T is said to be feasible for the initial condition (x,i) if there exists admissible policies u such that

$$T\, d_T = x + E\left\{\int_0^T u(x_u(t),\alpha(t))dt \;\middle|\; x(0)=x,\ \alpha(0)=i\right\} \tag{5.13}$$

we can write this using $\delta(\alpha(t),j)$ as

$$T\, d_T = x + \sum_{j\in S} E\left\{\int_0^T u(x_u(t),\alpha(t))\delta(\alpha(t),j)\ dt \;\middle|\; x(0)=x,\alpha(0)=i\right\} \tag{5.14}$$

We can define the proportion of time in the interval (0,T) that the system spends in machine state j given that the initial state is i as

$$\Pi_{ij}(T) = \frac{1}{T} E\left\{\int_0^T \delta(\alpha(t),j)\ dt \;\middle|\; \alpha(0) = i\right\} \tag{5.15}$$

Combining (5.13) - (5.15), the average demand rate d_T is said to be feasible if there is a control policy such that

$$T\, d_T = T \sum_{j\in S} \Pi_{ij}(T)\bar{u}_{ij}(x,T) + x \tag{5.16}$$

The equivalent production capacity set $\Phi_T(x,i)$ is defined as the set of d_T such that (5.16) is satisfied. To verify that d_T is in $\Phi_T(x,i)$, it is sufficient to find $u_j \in \Omega(j)$ for each machine state such that

$$T\, d_T = T \sum_{j \in S} \Pi_{ij}(T) u_j + x \qquad (5.17)$$

Given a set of flow rates u_j , an admissible policy with $\bar{u}_{ij}(x,T)=u_j$ can always be found (one example is $u(x,i,t) = u_i \ \forall\ x,i,t$). Once the probabilities $\Pi_{ij}(T)$ are evaluated, the solution of (5.17) is fairly simple. Workcenter managers are likely to know the initial state of the system and so they might use (5.17) in their production planning process. However, the calculation of $\Pi_{ij}(T)$ may be difficult. The steady state form (5.8)-(5.9) may be more appropriate for higher levels of management which have longer planning horizons.

Because $\Phi_T(x,i)$ is the set of expected production rates, it is possible that the actual production on a given day will not lie in $\Phi_T(x,i)$. Thus a demand rate d_T not in $\Phi_T(x,i)$ can be satisfied if the number of failures is less than expected. On the other hand a day with more failures than usual, could result in a demand d_T within $\Phi_T(x,i)$ not being satisfied. The finite time equivalent production capacity set $\Phi_T(x,i)$ therefore provides a useful planning tool but does not guarantee that demand will be satisfied.

5.3 A Production Planning Scenario Using the Equivalent Production Constraint Set

The equivalent production constraint sets Φ and $\Phi_T(x,i)$ describe the capacity of the manufacturing system. Given any production requirement, equations (5.8) and (5.9) or (5.17), which are a set of linear inequalities, determine whether or not it is within the capacity of the system. The choice of demand rates d(t) to impose on the hierarchical controller depend on the

type of manufacturing system involved. However, we can describe a scenario which applies to the system described in Chapter VI.

The facility managers have to satisfy a total demand Q_w for a part family over a period of a week. The quantity Q_w is determined by a requirement plan by higher management levels. The task is to choose daily production quantities q_k $k = 1,2,\ldots,5$ to set as targets for the workcenter controller. One choice assuming a five day working week is

$$q_k = \frac{1}{5} Q_w \qquad \forall k.$$

It may be advantageous, however, to produce a different mix of parts each day. For example, if there is a reward $c_k' q_k$ associated with producing quantity q_k on day k and T_k is the number of hours to be worked on day k, the following linear program is appropriate.

$$\max \quad \sum_{k=1}^{5} c_k' q_k \tag{5.18}$$

$$\text{subject to} \quad \sum_{k=1}^{5} q_k \geq Q_w \tag{5.19}$$

$$\text{and} \qquad q_k/T_k \in \Phi \ \forall\, k \tag{5.20}$$

The solution to (5.18) - (5.20) is a sequence of daily production targets which are at extreme points of Φ. If there is no feasible solution, then the production target Q_w cannot be met with the resources available.

The production planning problem sketched above falls into the framework

of multi-product production and inventory control theory. An extensive body of literature exists describing many different variations both static and dynamic of the problem. A survey is given by Aggarwal [1974]. Because the manufacturing system is represented by the deterministic equivalent production set Φ, efficient methods for deterministic planning problems may be employed.

The equivalent production constraint set Φ_T and $\Phi_T(x,i)$ depend on the parameters of the system. Section 2.4 discusses the need for parameter tracking algorithms in the overall production planning and control loop. Predictive algorithms determine whether or not schedules and control policies calculated using apriori estimates of system parameters are still feasible or will fully utilize the system during the horizon of control. For this reson, an open loop feedback approach [Bertsekas, 1976] to the solution of (5.18)-(5.20) is applicable. The sequence $\{q_k\}$ $k = 1,\ldots,5$ is evaluated initially using the best available parameter estimates. The first day's quota q_1 is given to the hierarchical controller for production, and the problem (5.18)-(5.20) is then resolved for the remaining production using again the best available parameter estimates. The process is repeated until the end of the week. An alternative approach is to solve (5.18) - (5.20) each day with a fixed horizon of several days. This method is termed rolling horizon production planning [Blackburn and Millen, 1980].

5.4 Summary

This chapter has given necessary and sufficient conditions for a demand requirement on a workcenter to be feasible. Simple procedures to verify the feasibility of a given demand rate are given and a production planning method for workcenter

managers is formulated. The exact form of the scheduling problem depends on the particular application.

The production planning process requires accurate knowledge of system parameters. The need to operate the system close to its estimated capacity makes the plan sensitive to errors in parameter estimates. Underestimating the capacity leads to low utilization of the system while overestimating the capacity results in a failure to meet production targets. The parameter tracking algorithms discussed in Section 2.4 should be integrated into the production planning scheme suggested in this chapter. For this reason, dynamic production planning such as is suggested in (5.18) - (5.20) is preferable to a static scheme. Our simulation results in Chapter VI show that it may not always be possible to satisfy demand rates that are close to the boundary of the equivalent production capacity due to modelling errors. The effects of modelling errors and unmodelled disturbances can be estimated by tracking the system buffer state and workstation utilizations. If there is evidence that the capacity of the system is inacurate, the set Φ or $\Phi_T(x,i)$ can be modified using the current parameter estimates and taking into account the lost production due to errors in the model and unmodelled disturbances.

CHAPTER VI

6.0 SIMULATION RESULTS

6.1 Introduction

The performance of the three-level hierarchical production control algorithm described in Chapter II is tested on a simulation model of a flexible assembly system. The model is based on a planned automated facility for assembling circuit boards.

Section 6.2 provides an overview of the manufacturing system. It consists of several flexible automated machines for inserting electronic components into printed circuit boards. The system is intended to produce a given quota of several different types of parts in a working day.

The simulation model is described briefly in Section 6.3 A detailed description can be found in Appendix III. The model keeps track of all individual workpieces within the system by assigning unique serial numbers as the parts are loaded. The code is modular allowing for different versions of the hierarchical controller to be easily tested. Both the estimate based (EB) controller of Section 4.3 and the certainty equivalent (CE) control algorithm of Section 4.4 are tested for the flow control level of the hierarchy. The sequence control level is implemented by the interval loading scheme described in Section 2.4. It determines the times at which to dispatch parts into the system by establishing earliest and latest times at which to load parts based on throughput rates chosen by the flow controller.

Two sets of simulation results are presented. Section 6.4 discusses results for a two-part three-stage system. The small system demonstrates important characteristics of the hierarchical control algorithm and provides insight into the behavior of the buffer state. In Section 6.5, a large four-

stage example is demonstrated. Here the emphasis is on the steady state characteristics of the control algorithm. The effect of modelling errors and parameter variations are tested on the simulated model.

6.2 The Manufacturing System

The circuit board assembly system is part of a factory that produces parts for computers and peripheral devices. The sub-system modelled here is the component insertion phase of the assembly process. It consists of machines that insert components such as resistors, capacitors and integrated circuit chips into printed circuit boards.

There are five machine types (A-E), which are grouped together in stages with the layout of Figure 6.1. Each type of machine is capable of handling one of five classes of circuit devices. For example, one class consists of axial lead components. Several different classes of type A and D machines are available. Some of these can only insert a subset of type A and D components. In some system configurations therefore, the parts may have to visit more than one type A and or D machine in order to receive all the necessary circuit components. Type B, C and E machines can handle all of their type of components. Thus each part need only visit one machine in these stages.

The machines all have a mean time between failures (MTBF) of 300 minutes and a mean time to repair (MTTR) of 30 minutes. In an average day, each machine can be expected to fail twice and be available for production 90% of the time.

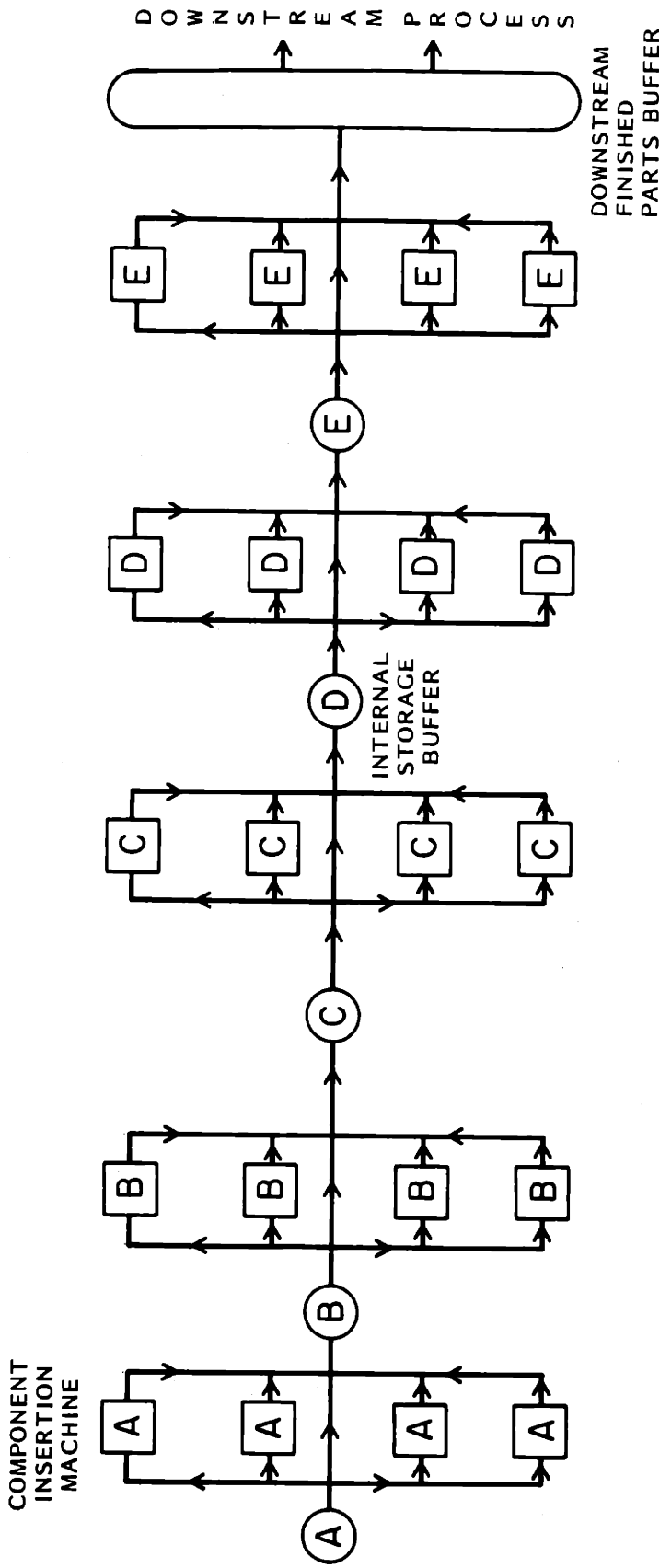

Figure 6.1 The Simulation Model of the Manufacturing System

The circuit cards are produced in lots with a typical day's total production between 5,000 and 10,000 parts. The number of different part types is large, but only a small subset is produced on any given day.

A day's production plan, in terms of quantity and part type, is decided in advance. The production scheduling task that is solved by the hierarchical algorithm is one of releasing work into the system so that the quota is met. There may be constraints as to the minimum number of parts of each type that can be loaded into the system together. The problem is then one of scheduling the release of the batches.

It is assumed that each day's production quota is within the capacity of the system. The managers of the system are therefore assumed to have used production planning tools such as those discussed in Chapter V to determine the daily parts requirements. This is especially important in a system in which the number of required part types is large.

To simplify the simulation study, we consider the production of four parts (1-4) and four stages (A-D) each with four machines. Type A and D machines are assumed to be able to insert all class A and D components respectively. In the simulation model therefore each part visits a single machine in each stage where circuit devices of that type are required.

Since the machines in each stage are assumed to be identical, the machine state is $\alpha(t) = (\alpha_A(t), \alpha_B(t), \alpha_C(t), \alpha_D(t))$ with the components defined as

$$\alpha_m = \text{number of operational machines in stage } m$$

Thus α_m takes integer values between 0 and M_m, the number of machines in stage m. The transitions of $\alpha(t)$ are then governed by

$$P(\alpha_m(t+\delta t) = j \mid \alpha_m(t)=i) = \begin{cases} ip_m\delta t+0(\delta t^2) & j=i-1,\ i > 0 \\ M_n - ir_m\delta t+0(\delta t^2) & j=i+1,\ i < M_m \\ 0 & \text{Otherwise} \end{cases}$$

Where i is a positive interger such that $0 \leq i \leq M_m$, p_m and r_m are the failure and repair rates of stage m machines respectively.

The time needed to mount and insert components for the parts at each stage are given in Table 6.1. The table also shows the proportion of each part in the daily requirement.

		PROCESSING TIME IN SECONDS				
PART	% OF TOTAL PRODUCTION	STAGE A	STAGE B	STAGE C	STAGE D	STAGE E
1	18	20	20	20	20	-
2	9	40	-	15	15	-
3	35	-	-	12	12	14
4	35	25	30	30	30	30
5	3	20	30	30	20	30

Table 6.1 Processing Time for Part Types 1-5 at Stages A-D in Seconds

6.3 The Simulation Model

The simulation is implemented by PL/1 subroutines that operate on a set of data structures describing the state of the simulated model. The simulation is run under the Honeywell Multics operating system at M.I.T.

Descriptions of the subroutines and the data structures are given in Appendix III.

An event-driven simulation is used. Each possible event in the system (for example an operation completion at a machine, a failure, or a repair) has an associated clock. The heart of the simulation is the EVENT MONITOR subroutine. It checks all clocks to determine the next event. The relevant action subroutine is called and all clocks are decremented by the time since the last event. The times between random events are produced by a random number generator with the appropriate probability distribution function.

When a failure occurs, a repair time is generated and loaded into the repair event clock for the failed machine. Similarly, when a repair is completed the time to the next failure is generated and loaded into the failure event clock.

The sequence controller, in addition to dispatching parts, has the task of making decisions internal to the system concerning individual pieces. For example, when an operation is completed, the sequence controller decides where the part goes next. Fixed internal decision rules are used. Internal storage buffers have a last-in, first-out discipline. This enables newly introduced parts to by-pass pieces already in the system so that the output flow rate quickly matches any new throughput rate determined by the flow controller.

Statistics on system performance are maintained for each part and machine in the system. They are updated after each event that occurs.

The simulation code is modular, allowing additional features and control and scheduling algorithms to be added with minimal modification.

6.4 Results for Two-Part Three-Stage Example

6.4.1 Introduction

Preliminary tests were carried out for a small subsection of the system consisting of stages A C and D and parts 1 and 2. The small system allows for a better intuitive understanding of the hierarchical production control algorithm because of the reduced number of parameters. The layout is shown in Figure 6.2 and parameter values are given in Table 6.2 and Table 6.3. The machines have a MTBF of 150 minutes and a MTTR of 30 resulting in an 83% availability for each machine. This is lower than the actual system. The more frequent failures allow for better observation of the effect of machine state changes on the algorithm.

The demand rate is set at 2.3 type 1 parts per minute and 1.15 parts per minute for type 2. The ratio requirement of Table 6.1 is thus satisfied. The equivalent steady state production capacity constraint set Φ is depicted in Figure 6.3. From the graph, the maximum production rate if the ratio of Table 6.1 is maintained is 2.45 and 1.22 parts per minute for type 1 and 2 respectively. The system is therefore required to operate at 94% of the effective capacity.

There are 27 possible machine states. Of the 27, only in 8 states is there at least one machine operational at each stage. The production constraint sets for these machine states and their relation to the demand rate are illustrated in Figure 6.4. The steady state probabilities of the productive states are given in Table 6.4. We note that the system

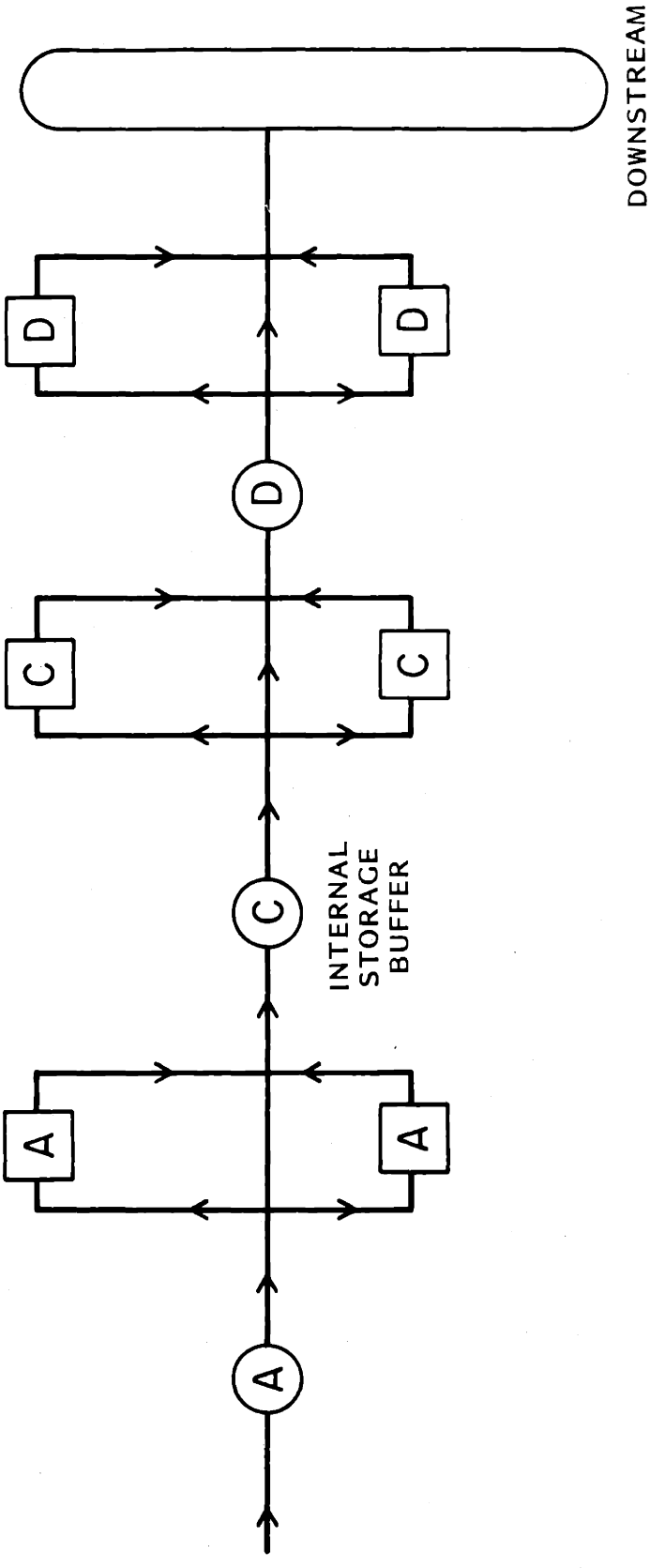

Figure 6.2 Three-Stage Subsystem

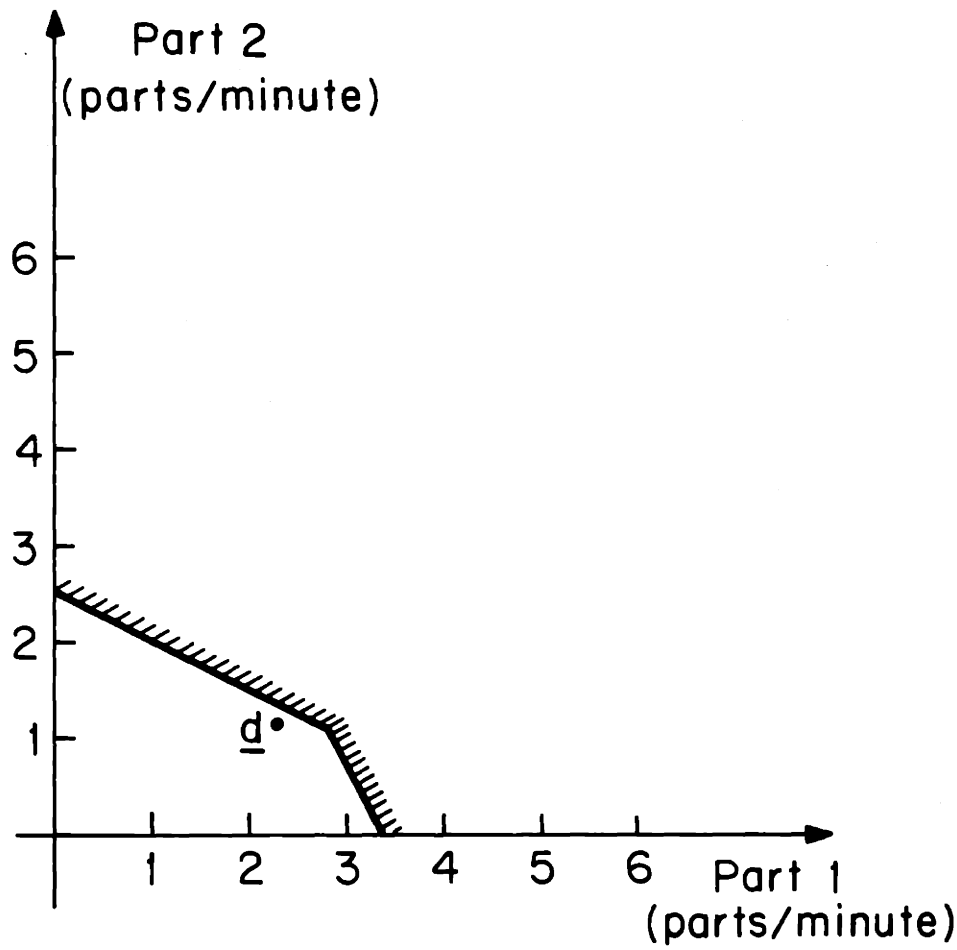

Figure 6.3 The Equivalent Production Capacity Set Φ for the Three-Stage Subsystem

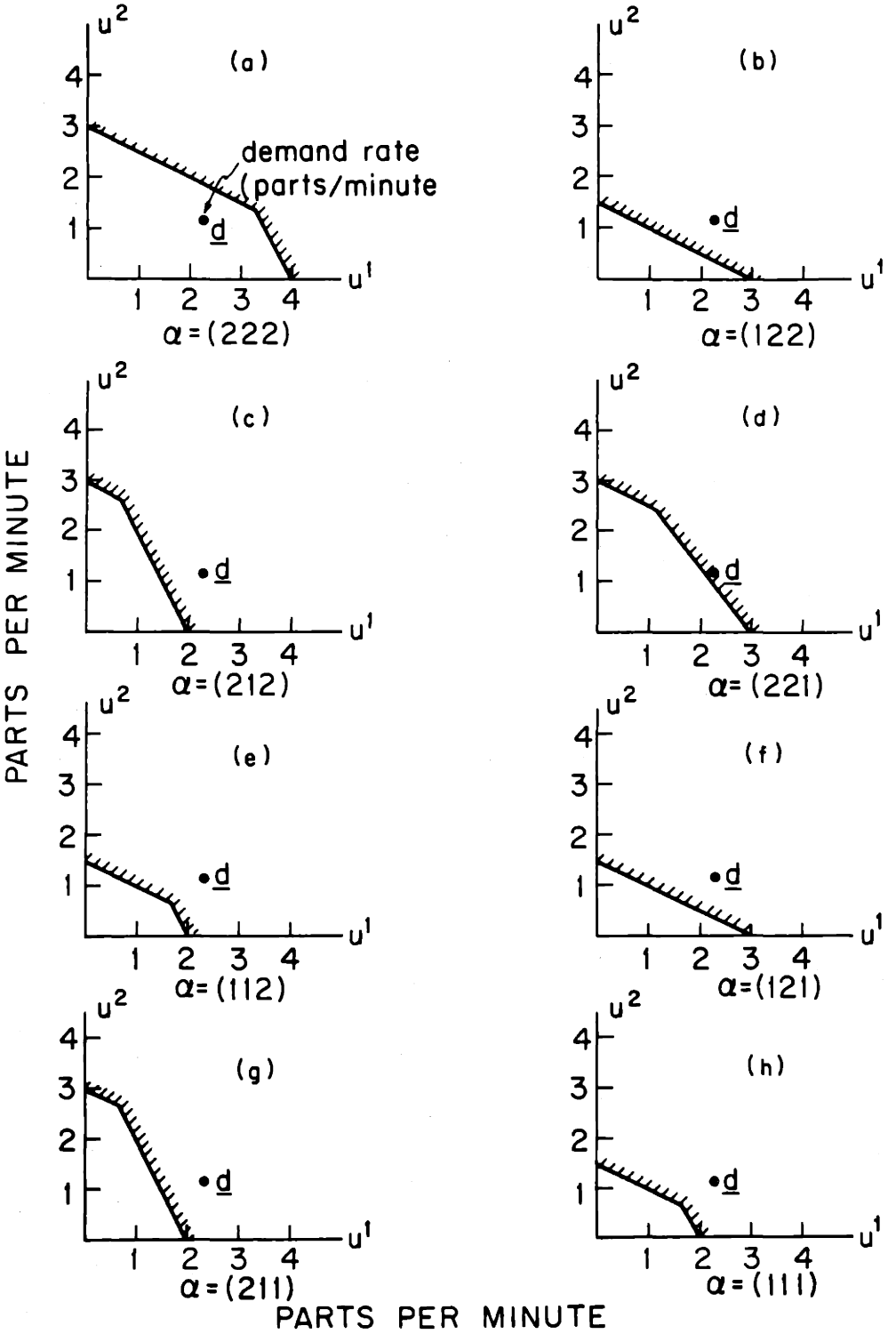

Figure 6.4 The Control Constraint Sets Ω for the Productive Machine States

PART	DEMAND RATE PARTS/MINUTE	PROCESSING TIME IN MINUTES STAGE A	STAGE C	STAGE D
1	2.3	.33	.5	.33
2	1.15	.67	.25	.25

Table 6.2 Demand Rates and Processing Times in Seconds for 2-part 3-stage Example

STAGE	NO. OF MACHINES	INTERNAL STORAGE BUFFER SIZE (No. of Parts)	MEAN TIME BETWEEN FAILURES (MTBF) Minutes	MEAN TIME TO REPAIR (MTTR) Minutes
A	2	1	150	30
C	2	1	150	30
D	2	1	150	30

Table 6.3 System Data for 2-part 3-stage Example

has all machines operating (α=(2,2,2)) only 33% of the time. Furthermore, only when all machines are operating can the system exceed the demand rate for both parts simultaneously. The most likely set of machine states includes those with one machine down which occur 40% of the time. Figures 6.4b - 6.4d show that the effect of a single failure on production varies with the stage in which the failure occurs.

The estimate based (EB) and certainty equivalent (CE) controllers of Section 4.3 and 4.4 respectively were tested for a fixed length simulation equivalent to 8 hours production.

MACHINE STATE α			PROPORTION OF TIME SPENT IN STATE (Steady State Probability)
A	C	D	
2	2	2	.33
1	2	2	.13
2	1	2	.13
1	1	2	.05
2	2	1	.13
1	2	1	.05
2	1	1	.05
1	1	1	.02

Table 6.4 Productive Machine States for 2-Part 3-Stage Example

In all test runs, the two controllers have the same sequence of failures and repairs. This is achieved by using the same set of seeds in the random number generator.

6.4.2 Results for the Estimate-Based Controller

The initial phase is the off-line compuation of the estimates $\Psi(x,i)$ of the cost-to-go function $J_{u^*}(x,i)$ by the successive approximations method described in Section 4.3.2. The cost index is given by

$$\lim_{T\to\infty} \frac{1}{T} E\left[\int_0^T |x^1(t)| + |x^2(t)|\right] dt \qquad (6.1)$$

The controller is thus penalized equally for being ahead or behind demand. Quadratic functions

$$\sum_{n=1}^{N} a_n^i (x^n)^2 + b_n^i x^n + c_n^i$$

are fitted to the estimates $\Psi(x,i)$ of the cost-to-go function for each machine state. This reduces storage requirements since we no longer have to store a large number of discrete data points. The on-line computation of the flow vector (4.20) for the EB controller is speeded up because numerical differentiation is avoided when the EB flow controller is activated on-line. In practice, the quadratic curves fit the numerically calculated estimates well. This is shown in Table 6.5 in which the coefficients of the quadratic functions and the mean square error between the fitted curve and the data points are presented for the upper bound $\bar{\Psi}(x,i)$ of the optimal cost-to-go function $J_{u^*}(x,i)$. For the simulation run, the upper bound was used as the estimate function. That is, $\eta = 0$ in equation (4.27).

The results of the simulation run are shown in Tables 6.6 and 6.7 for the EB flow controller. The availability of the machines is higher than the expected 83%. Thus this is equivalent to a lucky day when machine availability is higher than average.

STATE			UPPER BOUND						MEAN SQUARE ERROR	
α_1	α_2	α_3	a_1^i	a_2^i	b_1^i	b_2^i	c_1^i	c_2^i	PART 1	PART 2
2	2	2	1.6	3.2	-2.1	-10.8	0	0	1.1	2.4
1	2	2	1.1	2.5	-73.6	-95.6	2722	1795	228	631
2	1	2	1.2	2.6	-71.9	-92.5	2510	1646	258	671
1	1	2	0.9	2.2	-99.2	-125	4722	3080	353	1280
2	2	1	2.2	3.5	-51.5	-67.8	1456	939	795	1784
1	2	1	1.1	2.5	-95.3	-118	4227	2733	474	1292
2	1	1	1.2	2.7	-93.7	-115	3998	2576	538	1364
1	1	1	0.8	2.1	-113	-140	6134	3949	393	1525

Table 6.5 Quadratic Function Coefficients for the Upper Bound to the Cost-to-go Function for Productive States

STAGE	AVAILABILITY	UTILIZATION
A	.88	.86
C	.96	.73
D	.98	.53

Table 6.6 Machine Availability and Utilization for the EB Controller in the 3-stage 2-Part Example

PART	REQUIREMENT	PRODUCTION	MEAN BUFFER STATE	MEAN PROCESS INVENTORY	MEAN (Min) TRANSIT TIME
1	1104	1095	-8.2	3.4	1.5
2	552	538	-4.2	1.7	1.5

Table 6.7 EB Controller Production Statistics for the 3-Stage 2-Part Example

During the simulation run, the production vector u(t) was chosen at one minute intervals using the linear program (4.20) and the stored estimates $\Psi(x,i)$. The solution of (4.20) for this example is very simple and requires less than 1 second of C.P.U. time on the Honeywell computer. The sequence control scheme implemented in the simulation verifies whether or not the flow rate for a part has changed before establishing new loading windows.

Figure 6.5 is a plot of the buffer state x(t) trajectory plotted at one minute intervals for the EB controller. The buffer state stays at the hedging point which is approximately (5.5).

When failures occur, production falls behind demand because for any machine state with a failed machine, the demand rate is outside the capacity constraint set as is shown in Figure 6.4. After the repair is completed, the buffer state recovers to the hedging point. Figure 6.6 depicts a portion of the trajectory superimposed on the control regions for machine states $\alpha = (2,2,2)$ and $\alpha = (2,1,2)$. These are the states with all machines up and with one stage C machine failed respectively. The buffer state is at the hedging point and the machine state is (2,2,2)

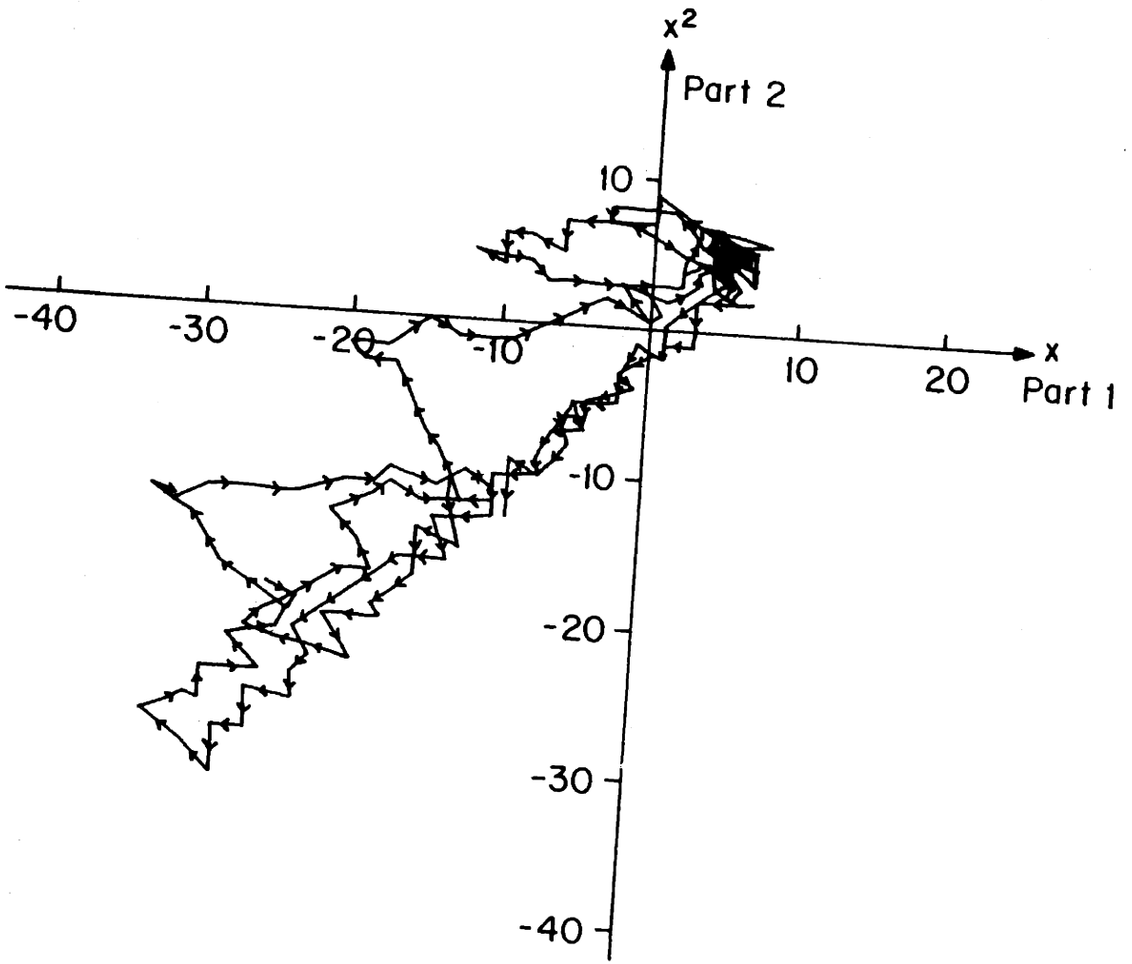

Figure 6.5 The Buffer State Trajectory for the Three-Stage Two-Part Subsystem with EB Flow Control Algorithm

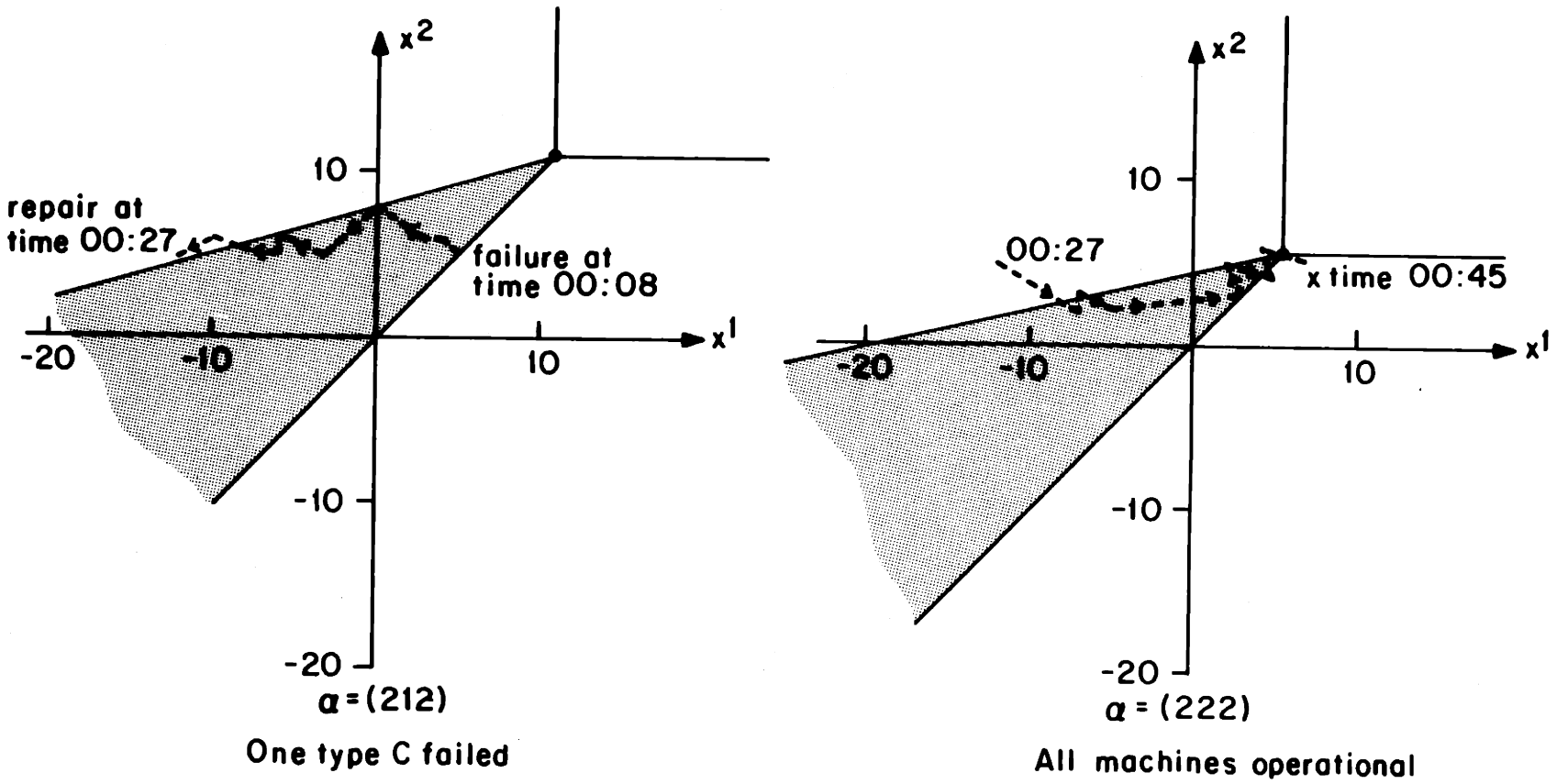

Figure 6.6 A Section of the Trajectory of Figure 6.5 Superimposed on the Control Regions for the Three-Stage Two-Part Subsystem

when a stage C machine fails at time 00hrs:08 mins. The control $u = (.667,2.667)$ which is an extreme point of the constraint set of Figure 6.4c, is applied. The trajectory hits the boundary at the point $x = (0,8)$ and control switches to the point $u = (2,0)$. There after, it switches between $u = (2,0)$ and $u = (.667,2.667)$ keeping the trajectory along the boundary. After 19 minutes, the failed machine is repaired. The control $u = (4,0)$, an extreme point of Figure 6.4a, is applied, driving the trajectory to the boundary at the point $x = (7,2)$. The control $u = (3.333,1.33)$ is applied. There is a momentary switch back to $u = (4,0)$ before the control $u=(3.333,1.333)$ is applied to drive the trajectory to the boundary at $x = (3,3)$. There are several switches of the control before the trajectory reaches the hedging point after approximately 45 minutes.

The implementation of the flow controller in the simulation model determines the control vector approximately every minute. No attempt is made to detect the boundaries of the control regions. This leads to the switching of the control between adjacent extreme points when the buffer state is close to a boundary. The performance of the system is not adversely affected by the switching and it does not seem worth the computational cost to implement boundary detecting code in the simulation.

6.4.3 Results for the Certainty Equivalent Controller

The short test is repeated for the CE controller of Section 4.4. At one minute intervals, the mathematical programming problem (4.64)-(4.65) is solved with

$$g(x) = g_{T_i}(x) = \max_n |x^n - y^n| \qquad (6.2)$$

The CE controller's objective is thus to minimize the maximum deviation from the hedging stock y. Machine availability and utilization of available time and final production statistics are shown in Tables 6.8 and 6.9 for y = 0 and y = 10. The corresponding buffer state trajectories are shown in Figure 6.7 for y = 0 and Figure 6.8 for y = 10.

With y = 0 the controller has a hedging point at about the x = (8,7) point. Thus the CE controller exhibits hedging behavior. For y = 10, the hedging point is x = (20,16). The trajectories of Figure 6.7 and 6.8 are very similar except for the translation due to different hedging points.

STAGE	AVAILABILITY	UTILIZATION	
		y = 0	y = 10
A	.88	.87	.88
C	.96	.74	.75
D	.98	.54	.54

Table 6.8 Machine Availability and Utilization for the CE Controller in the 3-Stage 2-Part Example

PART	REQUIREMENT	PRODUCTION		MEAN BUFFER STATE		MEAN INPROCESS INVENTORY		MEAN TRANSIT TIME	
		y = 0	y = 10	y = 0	y = 10	y = 0	y = 10	y = 0	y = 10
1	1104	1108	1114	1.2	6.5	3.6	3.7	1.55	1.6
2	552	536	545	-6.2	-1.9	1.8	1.9	1.6	1.7

Table 6.9 CE Controller Production Statistics for the 3-Stage 2-Part Example

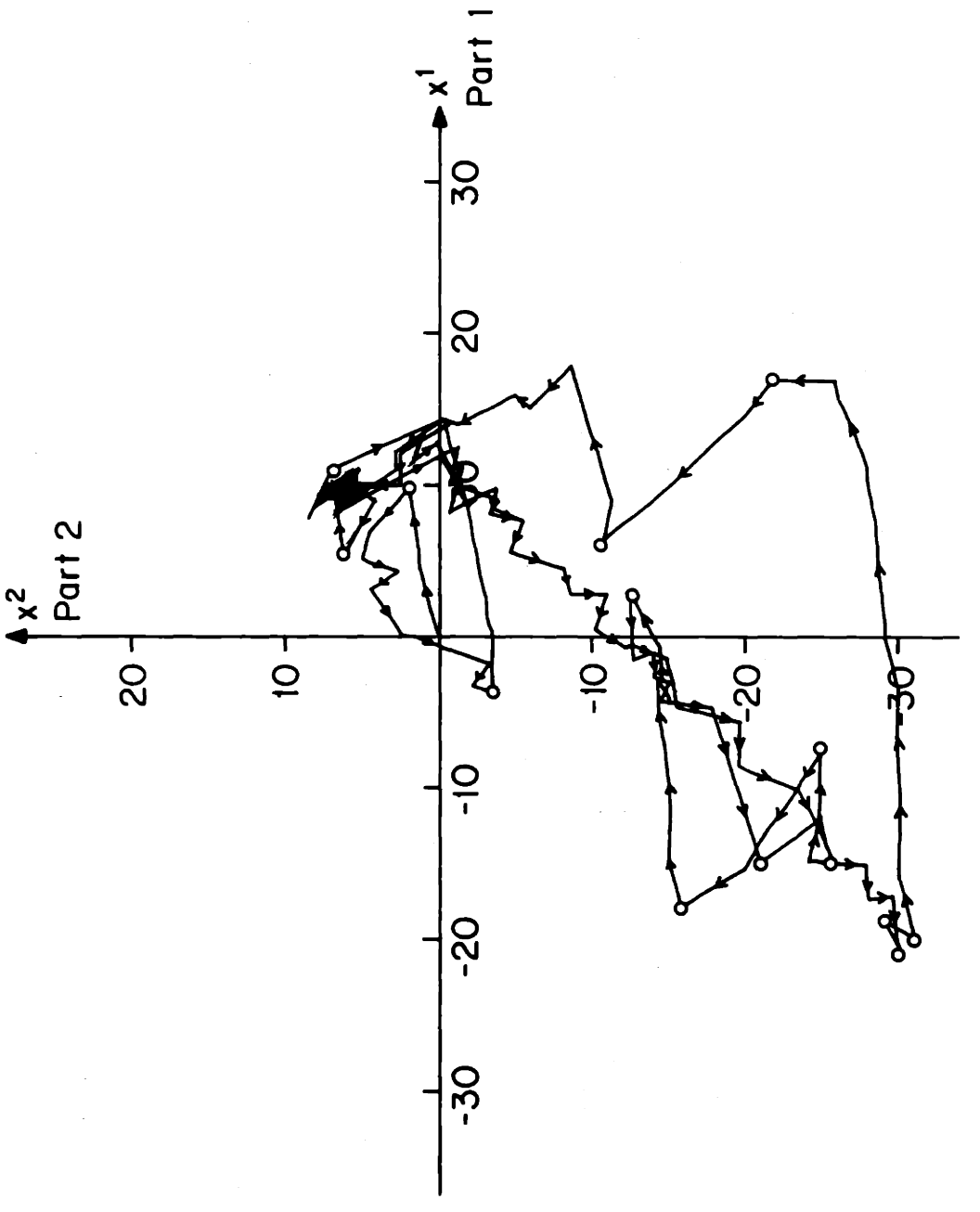

Figure 6.7 The Buffer State Trajectory for the Three-Stage Two-Part Example Using the CE Controller with Buffer Stock y = 0

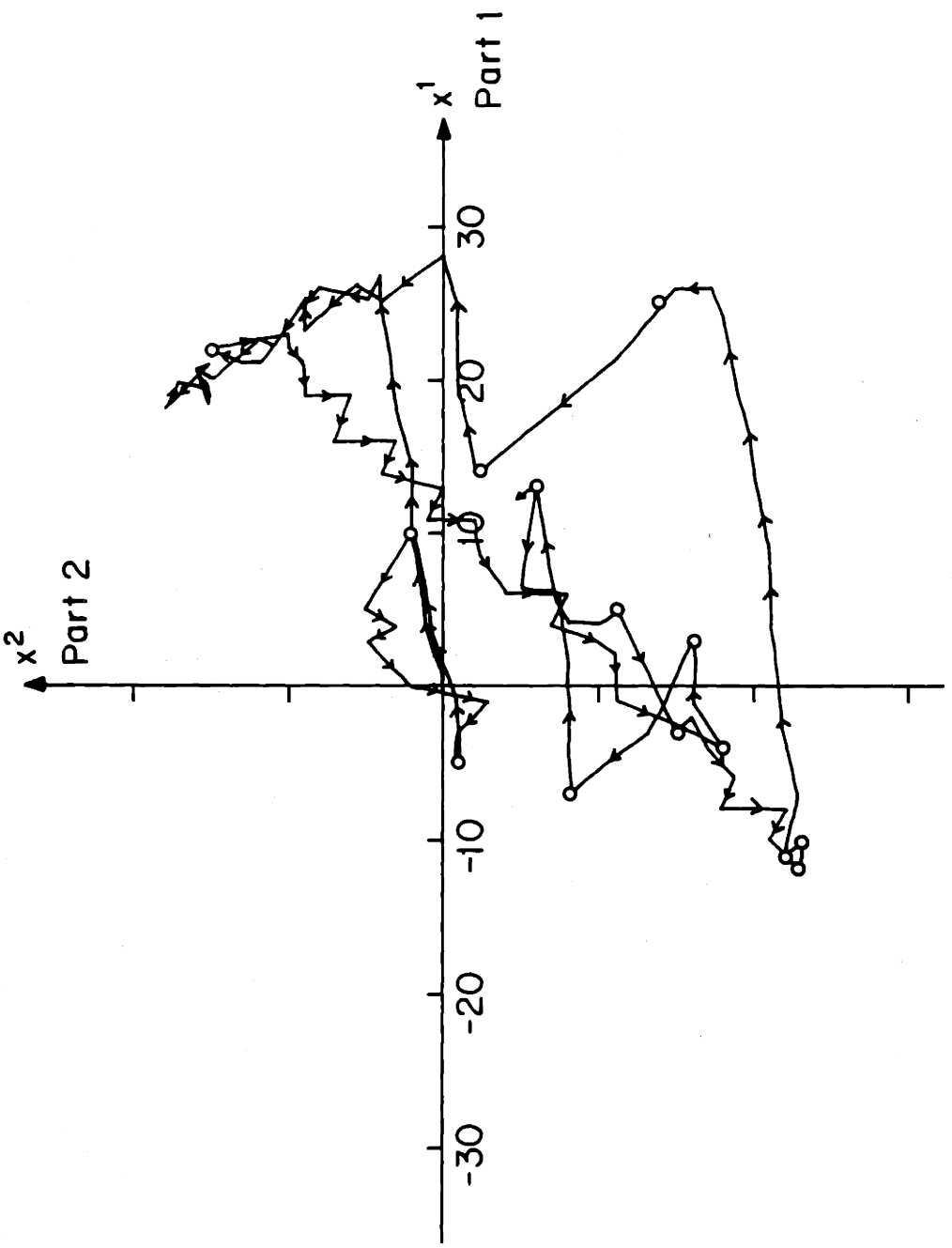

Figure 6.8 The Buffer State Trajectory for the Three-Stage Two-Part Example for the CE Controller with Buffer Stock y = 10

6.4.4 Characteristics of Estimate Based and Certainty Equivalent Flow Controllers

Comparison of the EB and CE controller trajectories of Figures 6.4 and 6.6-6.7 respectively show that the CE controller tends to favor the production of type 1 over type 2 parts. The EB controller keeps the backlogs or inventories of both the parts close together, that is close to the line $x^1 = x^2$. It would appear that the CE controller takes mainly into account the fact that type 1 parts have a higher demand than type 2 parts and ignores the higher capacity of the system to produce type 1 parts. The EB controller accounts for this by having the estimated cost-to-go function penalize backlogs of type 2 parts more than that of type 1 parts as is shown in Table 6.5.

The short tests show some important characteristics of the hierarchical controller. The buffer state responds quickly to the control applied by the flow control algorithm. The interval loading scheme appears to be effective in producing the flow rates determined by the higher levels when these flow rates are within the control constraint set.

Both versions of the flow controllers come very close to the production target. The EB controller is 9 type 1 and 12 type 2 pieces short of target at the end of the simulation run while with y = 0, the CE controller is 4 type 1 pieces over the target and 16 type 2 parts short at the end of the simulation run. The average number of pieces inside the workcenter (the mean in-process inventory), is small for

both flow controllers. The mean transit time is the total amount of time that the average piece spends inside the system. This includes both waiting and operation time. Thus for example, from Table 6.7, the average piece spends 22% of the time inside the workcenter waiting in internal storage buffers when the EB controller is used. The CE controller has approximately the same proportion of waiting time.

Initially when the simulation was implemented, there were instances when the interval loading scheme failed to maintain the chosen flow rates. This was because whenever a load request issued by the sequence controller could not be satisfied, it was placed in a stack of load requests ordered according to the latest time to load. The subroutine LOAD-PART which loads parts into the system always starts at the top of the stack to pick the part with the earliest deadline, and loads pieces into the system as soon as a space appears in the system. Normally, the stack of pending load requests contains very few pieces since load requests are generated by the sequence controller at a rate equal to the flow rate determined by the flow control algorithm.

When a workstation failed with pending load requests unsatisfied, the flow controller responds to failures by reducing the input flow rate and changing the part mix. However, if there are pending load requests, the sequence control algorithm still tries to satisfy them as soon as possible. The result is that parts enter the system at a rate which is different from that set by the flow controller and higher than the current capacity of the system. This results in blocked stations and a production rate different from the chosen control vector u(t). By

canceling pending load requests after a machine state change the build-up of a large queue of load requests is avoided. The sequence controller is then more responsive to the control set by the flow control algorithm.

The effect of canceling load requests is demonstrated in Figures 6.9 and 6.10 which show the buffer state trajectory only after machine state changes. Figure 6.9 corresponds to the trajectory of Figure 6.5 and has the cancel load request feature. Figure 6.10 is the result of running the simulation with the same sequence of failures and repairs but without canceling pending load requests after failures and repairs. The trajectory of Figure 6.10 does not stay as close to the $x^2 = x^1$ line as that of Figure 6.9. This is because not canceling pending load requests results in a mismatch between desired and actual input flow rates. Note that while the trajectory of Figure 6.9 recovers to the hedging point x = (5,5) that of Figure 6.10 does not.

The behavior of the hierarchical EB controller is determined by the unconstrained minimum $\hat{x}_i$ of the estimates to the cost-to-go functions $\Psi(x,i)$. Away from $\hat{x}_i$ and when d is within the constraint set $\Omega(i)$, equation (4.54), which determines the control, picks u(x,i) such that u(x,i)-d is the steepest descent direction within the constraint set $\Omega(i)$ in the minimization of $\Psi(x,i)$. Otherwise, if d is outside $\Omega(i)$, u(x,i)-d is the direction that results in the smallest increase in $\Psi(x,i)$. The control thus depends on the shape of the level surfaces of $\Psi(x,i)$ and not on its value.

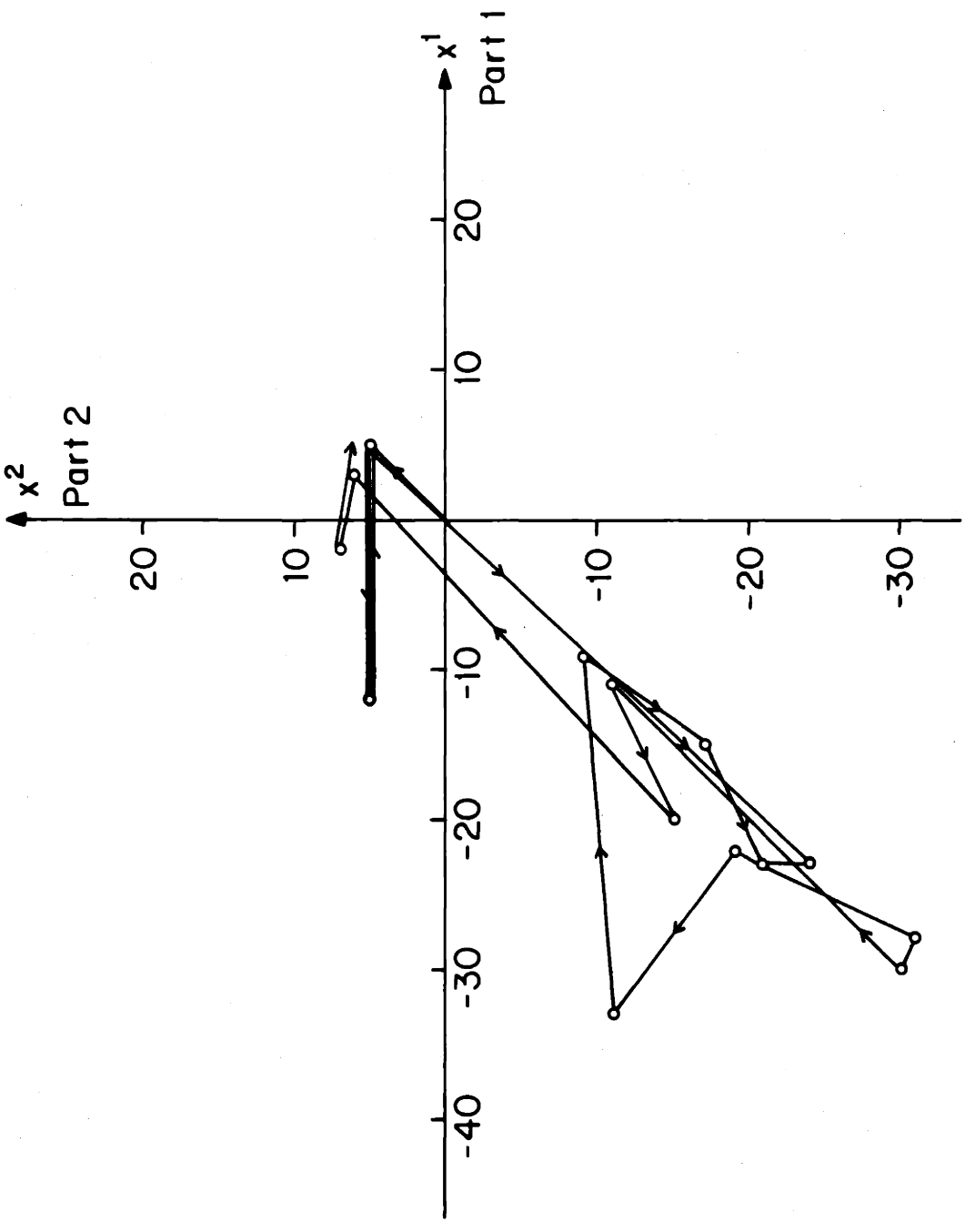

Figure 6.9 The Buffer State Trajectory for the Three-Stage Two-Part Example With Load Requests Cancelled After Machine State Changes

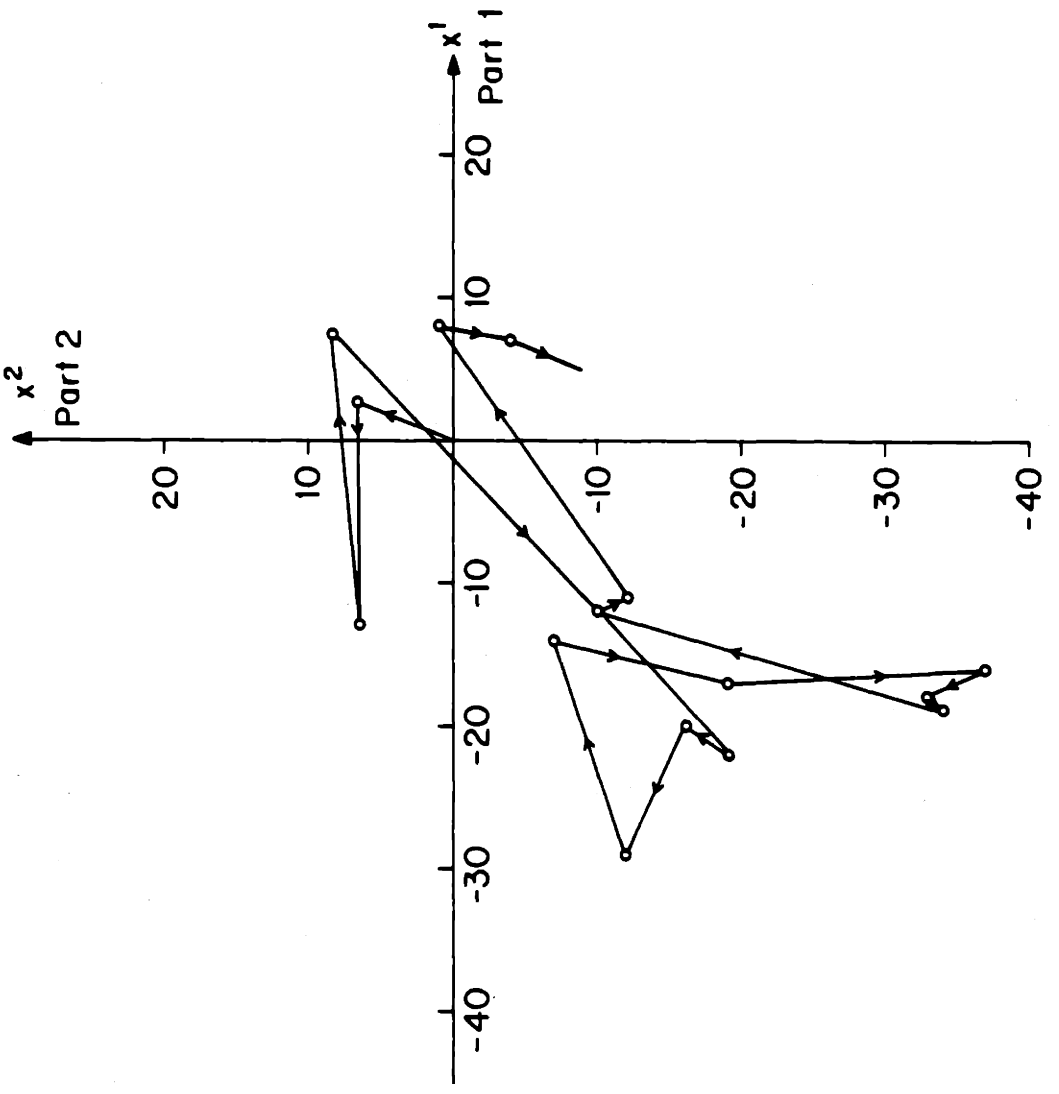

Figure 6.10 The Buffer State Trajectory for the Three-State Two-Part Example Without Cancelling Load Requests After Machine State Changes

The throughput rate chosen by a certainty equivalent control depends on the unconstrained minimum and the level surfaces of the objective functions g(x) and $g_{T_i}(x)$. In the example presented, the function minimized at each time t and buffer state x(t) is

$$\max_n \sum_j \lambda_{ij} \left| y^n - x^n(t) - u^n(t) - T_j u_j^n + (1+T_j) d_{tj}^n \right| \qquad (6.3)$$

The unconstrained minimum value of (6.3) is zero. When the buffer state is at the minimizing point $\hat{x}_i$ of (6.3) the solution to the minimization problem (6.3) is to produce at exactly the demand rate thereby keeping the buffer state at $\hat{x}_i$. It follows that $\hat{x}_i$ and u_j satisfy

$$y - \hat{x}_i - T_j(\hat{u}_j - d_{tj}) = 0 \qquad \forall\, j \in S \qquad (6.4)$$

The difference between the trajectories of Figures 6.7 and 6.8 can be explained in terms of the different level surfaces of the cost functions (6.2) and the locations of their unconstrained minimum for different values of y in equation (6.3).

6.4.5 Conclusion

The three level hierarchical production control algorithm effectively meets production targets despite the failures and repairs of the workstations. The behavior of the buffer state trajectory

closely conforms to theoretical predictions even for short periods of time. This is because the interval loading scheme maintains throughput rates that are close to the rates set by the flow control level of the hierarchy.

The estimate based flow controller tracks the demand rate more accurately than the certainty equivalent controller. This is evident from the mean buffer state and inprocess inventory shown in Tables 6.7 and 6.9, and the buffer state trajectories of Figures 6.5, 6.7 and 6.10. Thus the off-line computation for the EB flow controller appears to take sufficient account of the relative demand rates, productive capacity and value in the cost function for the two parts.

In the next section, we investigate the performance of the hierarchical controller on a larger system and over longer intervals of time.

6.5 Four-Stage Four-Part Example

6.5.1 Production Planning

The system tested in this Section consists of stages A-D each with 4 machines and parts 1-4. System capacity and reliability data are shown on Table 6.10. The processing times at each stage are given in Table 6.1. The required proportions of each part in the total production (the part production ratios) are adjusted and shown in Table 6.11.

The initial phase of running the simulation is production planning.Since the part production ratios are specified, we have to determine the total number of parts to be produced in a day.

The machines are assumed to fail and be repaired independently. For production planning purposes, each stage can therefore be replaced by the expected number of machines $\bar{M}$ at the stage. The times between failure and to repair are independent exponentially distributed random variables. The expected number of machines $\bar{M}$ is therefore given by [Kleinrock, 1975]

$$\bar{M} = \frac{rM}{r+p} \tag{6.5}$$

where M is the number of machines at the stage, 1/r the mean time to repair and 1/p the mean time to fail. With the data of Table 6.10, $\bar{M}$ = 3.64 for all stages. Using the part ratios of Table 6.11 and the processing times of Table 6.1, the total number of parts that each stage can handle in a 16 hour day is shown in Table 6.10. Table 6.11 shows the resulting daily requirement for each part and the throughput rate that the hierarchical controller must attain in order to satisfy the demand.

From Tables 6.10 and 6.11, the production target is within 0.2% of the effective capacity of the bottleneck stages C and D. The demand rate is therefore very close to the boundary of the equivalent production constraint set Φ. The controller should therefore utilize 100% of the available time at stages C and D. Stage A and B have a higher capacity and can be expected to be idle (that is, not failed and not working) 19% and 25% of the time respectively. At this point in the

production planning process, additional parts that require only components inserted at stages A and B could be added to the production requirement thereby using the spare capacity available at the first two stages.

Once production planning is completed and the demand rates determined, the next step is the calculation of the estimates $\Psi(x,i)$ to the cost-to-go functions. There are 625 machine states. However, if we exclude all states in which four machines are failed at any stage, that leaves 256 machine states in which estimates must be evaluated.

STAGE	A	B	C	D
TOTAL CAPACITY WITH SPECIFIED PART RATIOS, MTBF AND MTTR; PARTS PER 16 HOUR DAY	12700	14300	10300	10300
EXPECTED UTILIZATION OF AVAILABLE TIME WITH REQUIREMENTS OF TABLE 6.8 (%)	81%	72%	100%	100%
MEAN TIME TO FAIL (MINUTES)	300	300	300	300
MEAN TIME TO REPAIR (MINUTES)	30	30	30	30
NUMBER OF MACHINES	4	4	4	4

Table 6.10 System Capacity and Reliability Data for 4-Part 4-Stage Example

PART	DAILY REQUIREMENT	DEMAND RATE PARTS/MINUTE	RATIO OF TOTAL PRODUCTION (%)
1	1949	2.03	19
2	922	.96	9
3	3706	3.86	36
4	3706	3.86	36
TOTAL	10283	10.71	100

Table 6.11 Production Requirement for 4-Part 4-Stage Example

The upper bound $\bar{\Psi}(x,i)$ of the cost-to-go functions $J_{u^*}(x,i)$ was computed. Quadratic functions

$$\xi_n(x) = \sum_n a_n^i (x^n)^2 + b_n^i x^n + c_n^i \qquad (6.6)$$

were fitted to the numerical values of the $\bar{\Psi}(x,i)$. The coefficients of the quadratic functions are shown in Table 6.12 for machine states with all machines operational and one failed. The hedging points x_i^* induced by the quadratic functions are also given. The calculation of the estimates required 1.3 hours of C.P.U. time on a Honeywell 640 computer with the MULTICS time sharing operating system.

MACHINE STATE	a_1^i	a_2^i	a_3^i	a_4^i	b_1^i	b_2^i	b_3^i	b_4^i	x_i^{1*}	x_i^{2*}	x_i^{3*}	x_i^{4*}
4 4 4 4	6.9	3.5	6.9	8.5	-593	-803	-288	-315	43	114	21	7
3 4 4 4	4.9	2.4	6.9	6.5	-775	-895	-288	-559	79	186	21	43
4 3 4 4	6.7	3.5	6.9	8.3	-612	-803	-288	-341	46	114	21	20
4 4 3 4	5.2	4.7	6.6	9.2	-845	-797	-521	-555	81	85	39	30
4 4 4 3	5.2	4.7	6.6	9.2	-824	-797	-520	-555	79	85	39	30

Table 6.12 Quadratic Fit to Estimate of Cost-to-go Function Showing Induced Hedging Points x_i^* for the Most Probable Machine States

The steady state probabilities and mean soujourn times for the most likely machine states are given in Table 6.13. The system spends 78% of the time with at least one failed machine. We note also that 4-6 machine state changes can be expected every hour.

6.5.2 Simulation Results for the 4-Stage 4-Part Example

The system described by Table 6.10 with the part requirement of Table 6.11 was simulated for the equivalent of 16 hours. The length of the simulation run is similar to a day's production period and is long enough to allow many failure and repair events to take place. The capacity of internal storage buffers at each stage is one piece. Table 6.14 shows the availability and utilization of available time at each stage. The randomly generated failures and repairs result in availabilities that are within 4% of the expected 90%.

The hierarchical controller does not fully utilize all the available time at the bottleneck stages. The effect of the lower than expected utilization is evident in the production figures of Table 6.15. The total number of parts produced is 9095 which is 88% of the required 10300 pieces. This proportion is close to the utilization of available time attained by the controller.

Figure 6.11 is an hourly plot of the buffer state for the four parts. It is clear early in the run that the system cannot keep up with the demand rate. We note that even though the system fails to meet the demand, the part production ratio at the end of the run is within 3% of the desired ratios.

NO. OF MACHINE FAILED	STEADY STATE PROBABILITY	EXPECTED TIME IN STATE (MINUTES)
0	.2176	15
1	.3482	12
2	.2612	12
3	.1219	12
TOTAL	.9489	

Table 6.13 Steady State Probabilities for the Most Likely System States in the 4-Part 4-Stage Example

STAGE	AVAILABILITY (%)	UTILIZATION (%)
A	94%	64%
B	91	63
C	87	92
D	93	86

Table 6.14 Workstation Utilization and Availability With Buffers Size of 1 With Demand Requirement of Table 6.10

PART	PARTS PRODUCED	% OF TOTAL PRODUCTION	FINAL BUFFER STATE	MEAN BUFFER STATE	MEAN INPROCESS INVENTORY	AVERAGE TRANSIT TIME (MIN)	% OF TIME ACTUALLY BEING OPERATED ON
1	1637	18	-312	-181	3.3	1.94	81.3
2	589	6.5	-333	-175	1.0	1.62	72.0
3	3586	39.4	-120	-91	2.1	.56	71.4
4	3283	36.1	-423	-265	9.0	2.6	73.7
TOTAL	9095	100	-1188	-712	15.4	1.6	74.1

Table 6.15 Part production with Buffers of Size 1
With Demand Requirement of Table 6.10

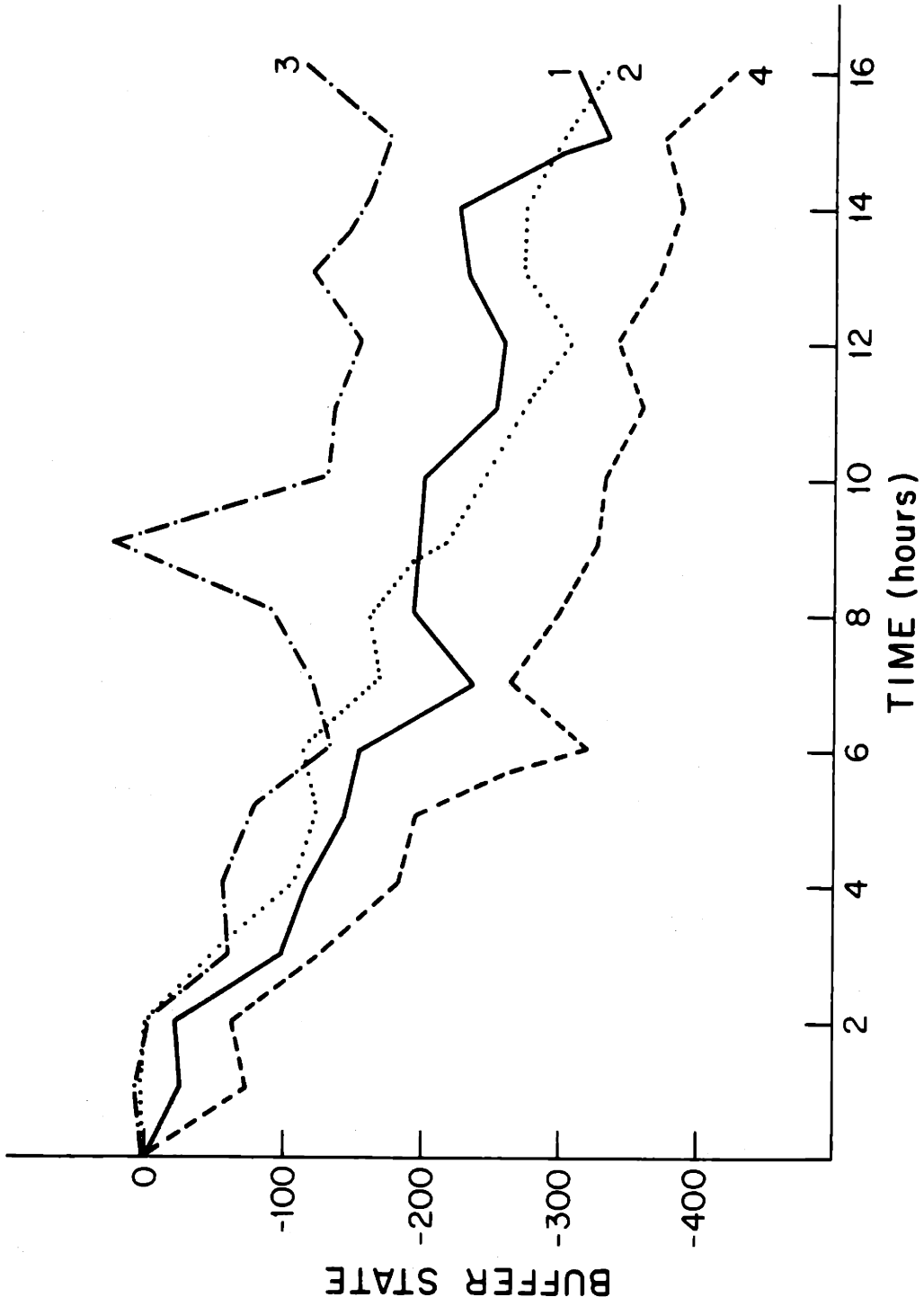

Figure 6.11 Buffer Levels for the Four-Stage Four-Part Example With Demand Rate of Table 6.11

Figure 6.12 is a plot of the buffer and machine states at one minute intervals during the 8th hour of the simulation run. The graph shows the effect of machine state changes on the throughput rates (i.e., the slopes of the curves) computed by the flow controller. Frequent changes of the control vector are evident.

From Table 6.15 the average number of pieces within the system during the simulation run is 15.4, with type 4 parts accounting for the major proportion. This is not surprising since type 4 parts have a high demand rate and require components from every stage. From the mean production rate and inprocess inventory, we can deduce using Little's formula [Kleinrock, 1975], that the average piece spends 1.6 minutes inside the system. The average transit times and ratio of total operation to transit time for all the parts is shown in Table 6.15.

To obtain high throughput rates while keeping inprocess inventory low, the proportion of time that parts spend waiting in queues while inside the system must be kept low. With internal buffers of size one, the hierarchical controller has a mean waiting time proportion of 26%. However, the utilization of available time at bottleneck stages is of the order of 90%.

The demand requirement of Table 6.11 was reduced by 5% to 9770 pieces while maintaining the part ratios. The simulation run was then repeated with internal buffer sizes of 1,5 and 10 with the same sequence of failures and repairs for each test. The results are displayed in Tables 6.16 - 6.18.

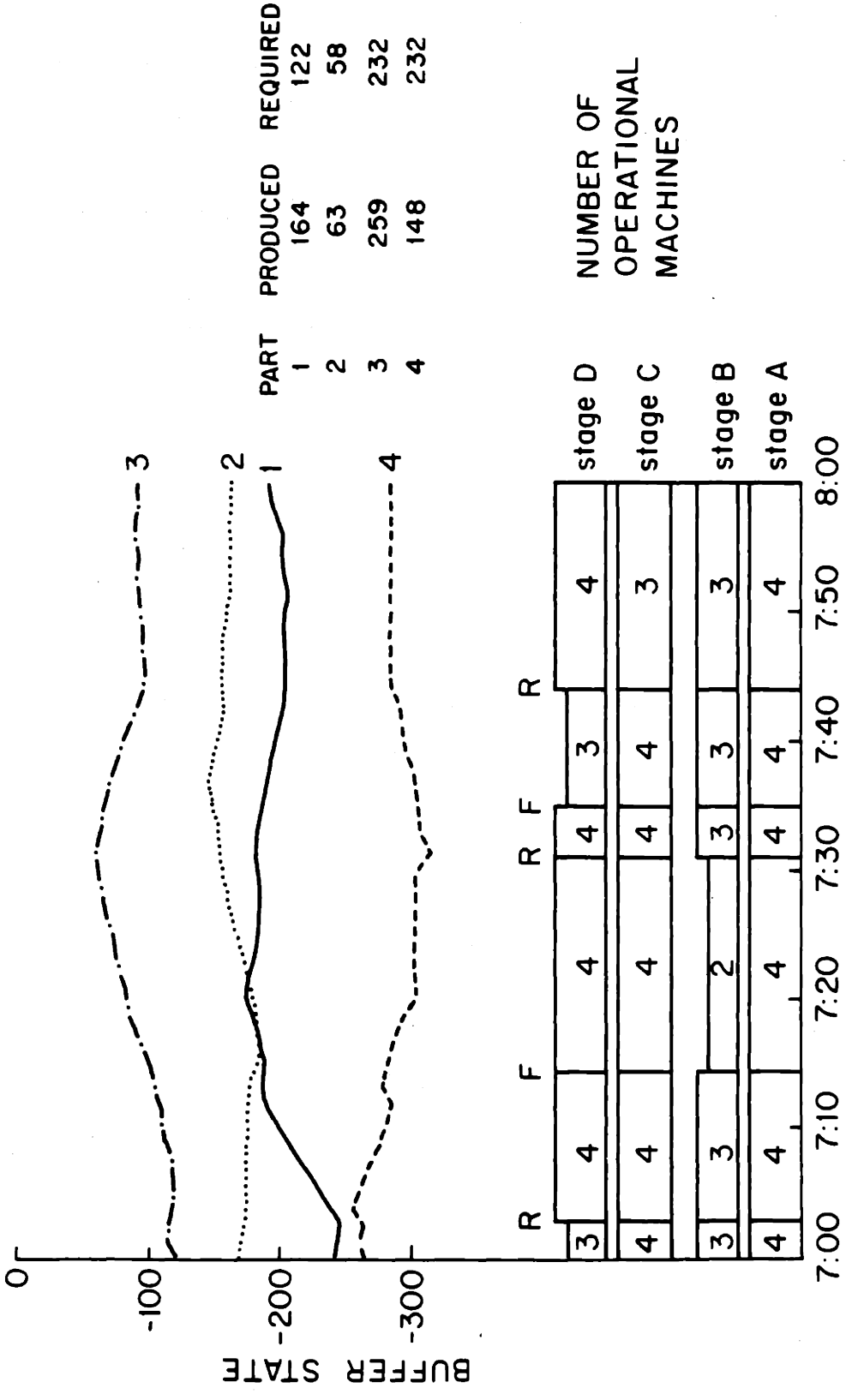

Figure 6.12 Buffer Levels and Machine States During the 8th hour of the Simulation Run

BUFFER SIZE	PARTS PRODUCED (% in total production) 1	2	3	4	TOTAL
1	1663 (18.4)	665 (7.4)	3446 (38.2	3257 (36.0	9031
5	1735 (18.6)	755 (8.1)	3460 (37.2)	3346 (36.0)	9296
10	1752 (18.7)	781 (8.3)	3483 (37.1)	3355 (36.0)	9371
REQUIREMENT	1852	876	3521	3521	9770

Table 6.16 Effect of Internal Buffer Sizes on Part Production

BUFFER SIZE	STAGE UTILIZATION (%) A	B	C	D
1	66	63	92	85
5	69	64	94	88
10	70	65	95	89

Table 6.17 Effect of Internal Buffer Sizes on Utilization

BUFFER SIZE	AVERAGE INPROCESS INVENTORY					MEAN TRANSIT TIME (MINUTES)
	1	2	3	4	TOTAL	
1	3.4	1.2	2.2	8.6	15.4	1.6
5	5.3	2.3	3.7	10.8	22.1	2.3
10	8.0	3.4	4.4	14.5	30.3	3.1

Table 6.18 Effect of Internal Buffer Sizes on the Average Number of Pieces Within the System

For buffers of size 1, Table 6.16 shows that the number of parts produced is reduced by 64 compared to the results displayed in Table 6.15. The part production ratios on the other hand are closer to the desired values. This is due to a higher production of type 1 and 2 and a reduced output of type 3 and 4 parts. The average inprocess inventory is unchanged.

Increasing the capacity of internal storages to 5 and 10 pieces leads to 2.9% and 3.8% improvement in throughput respectively. The part ratios also improve as does the utilization of the workstations. Table 6.18 shows that the mean inprocess inventory and transit times double when the buffer sizes are increased from 1 to 10 pieces. When internal storages have a capacity of 10 pieces, the avearge piece spends 3.1 minutes within the system 62% of which is waiting time in internal

buffers. Thus improving performance by enlarging internal storage is costly in terms of increased inprocess inventory.

We modified the sequence controller by having all load requests that were past due cancelled. In previous runs, as explained in Section 6.3, outstanding load requests were cancelled only after a failure or repair. The simulation run with the requirements of Table 6.16 and internal buffers of size 10 was repeated. The results are displayed in Table 6.19. Production decreases by 27 pieces which is negligible compared to the total output of 9344 parts. The mean inprocess inventory and transit times decrease by 15% and 16% respectively. We conclude that cancelation of overdue load requests is a useful feature to incorporate in the sequence controller. This also shows that dispatching pieces into the system as soon as a space appears does not always pay off since the pieces are likely to spend more of their time waiting in buffers. This results in a larger number of inprocess parts. In workcenters where the transportation system also serves as a temporary storage area, the additional congestion could also reduce the capacity of the transport mechanism. Higher inprocess inventories in palletized workcenters mean that a larger number of pallets is needed in order to achieve the required throughput.

In the simulation results presented above, the demand rate is beyond the capacity of the system due to the inability of the sequence controller to attain more than 90-93% utilization of bottleneck stages. The production target was reduced to 9254 pieces which is 90% of the original 10300. The simulation was run for the equivalent of four

PART	NO. PRODUCED	% OF TOTAL PRODUCTION	MEAN INPROCESS INVENTORY	MEAN TRANSIT TIME
1	1747	18.7	6.0	3.2
2	767	8.2	2.5	3.1
3	3473	37.1	5.6	1.5
4	3357	36.0	11.6	3.3
TOTAL	9344	100.0	25.7	2.6

Table 6.19 Effect of Canceling Past Due Load Requests with the Demand Requirement of Table 6.13

16-hour days with internal buffers with capacities of 10 pieces. Table 6.20 shows the production for the four runs without the cancelation of outstanding load requests described above. Run #1 is comparable to previous results because it has the same sequence of failures and repairs. We note that production is reduced by 46 pieces as compared to Table 6.16 which shows production with a higher requirement. The output shown in Table 6.20 is very close to the production target. This is due to an increased production of type 1 and 2 parts and a reduced production of types 3 and 4 parts. Runs 2-4 have different seeds for the random number generators so as to vary the failure and repair sequences. Production on all runs is consistent despite the variations in down times. Output on all runs exceeds the requirement because the controller tries to keep the buffer state at the hedging

PART	REQUIREMENT	SIMULATION RUN #1 1	2	3	4	MEAN DAILY PRODUCTION
1	1754	1787	1789	1784	1791	1788
2	830	888	878	875	946	897
3	3335	3338	3354	3344	3354	3346
4	3335	3312	3319	3309	3342	3320
TOTAL	9254	9325	9340	9312	9433	9352

Table 6.20 Daily Production When Demand Requirement is Within System Capacity

points of Table 6.12. Thus whenever possible, actual production is ahead of the demand by an amount that depends on the machine state and the hedging points.

Table 6.21 shows the average number of pieces within the system for the four runs. Again a comparison of run #1 with Table 6.15, shows that reducing the demand rate results in a slight decrease in the average number of pieces within the system. Run #4 shows a substantial decrease in the average number of pieces within the system. This is due to a higher than average availability at the bottleneck stages.

The sequence control algorithm is modified so that parts can visit the four stages in an arbitrary order. Whenever a part completes an operation, the next operation is performed at the stage with the least number of waiting parts. The part is sent to a central waiting area if all stages at which the parts requires an operation are occupied. These

AVERAGE INPROCESS INVENTORY					
	RUN #				MEAN INPROCESS
PART	1	2	3	4	INVENTORY
1	7.7	7.9	6.9	5.0	6.9
2	3.4	3.0	2.9	2.5	3.0
3	5.2	5.6	6.2	4.3	5.3
4	13.5	13.8	13.3	9.9	12.6
TOTAL	29.8	30.3	29.3	21.7	27.8

Table 6.21 Average Inprocess Inventory When Demand Rate is Within System Capacity

modifications reduce the possibility of blocked stations and the occurance of machines being starved of parts. Buzacott (1982) has also shown that central storage buffers result in higher throughputs for flexible manufacturing systems.

The simulation was run for the equivalent of 9.3 hours to test the effect of the modifications. Table 6.22 shows the workstation availability and utilization of available time. Compared to Table 6.17, the utilization of the workstations is improved with the exception of Stage C. We note that stage D which has a below average availability has a utilization of 97% compared with 89% before the modifications are made.

STAGE	AVAILABILITY (%)	UTILIZATION (%)
A	96	76
B	86	73
C	93	93
D	87	97

Table 6.22 Availability and Utilization of Available Time When Parts Visit the Stages in an Arbitrary Order and with a Large Central Storage Area

PART	AVERAGE NO. OF PIECES IN THE SYSTEM	MEAN TRANSIT TIME	PERCENTAGE OF TIME SPENT WAITING
1	13.4	6.6	80
2	8.9	8.2	86
3	57.6	15.5	97
4	30.9	8.4	77

Table 6.23 Effect of Large Central Storage Buffer on the Inprocess Inventory

The effect of the central storage area and arbitrary order of stage visits on the average number of pieces within the system is shown in Table 6.23. The average inprocess inventory increases dramatically compared to the bottom row of Table 6.18. In particular, we note that type 3 pieces spend 97% of the time inside the system waiting in storage. This is the same as the waiting time proportion in traditional job shops [Lerner, 1981] where it should be noted that workstation utilization is of the order of 5%.

Adding the central storage area shows that it is possible to increase the utilization of available time at bottleneck workstations beyond the 90% achieved by the simple interval loading scheme. However, a more sophisticated approach is necessary if the average number of pieces within the workcenter is to be kept small.

6.5.3 Conclusion

In this section, we have presented simulation results for a four-stage four-part system with 4 machines at each stage. The requirement on the system is stated in Table 6.1 as a ratio of each part in the total production of approximately 10000 pieces.

The initial phase of the simulation is production planning. Applying the equivalent capacity set Φ of Chapter 5, it is determined that the system is capable of producing 10300 pieces with the specified ratios in a 16 hour working day.

The simulation is run using the upper bound of the cost-to-go function as an estimate. The requirement for 10300 pieces is not met because

the interval loading scheme is only able to achieve approximately 90% utilization of the available time at the bottleneck stages. The calculation of the equivalent capacity set assumes 100% utilization. Even though part requirement is beyond the actual capacity of the system, the control algorithm maintains the part ratios within 3% of the required amount. The buffer state is found to be responsive to the chosen flow rate u(t) as is shown in the graphs of Figures 6.5 and 6.6 .

The part requirement is reduced by 5% to 9770 pieces and the simulation run is repeated with internal buffer sizes of different capacities. Increasing the capacity of internal buffers increases the utilization of available time at the bottleneck workstations with a consequent increase in total production. The average inprocess inventory on the other hand increases but at a much faster rate. Cancelling overdue load requests reduces inprocess inventory by 15% with a negligible decrease in throughput. This demonstrates the advantage of the flow control algorithm over any scheme that dispatches parts into the system whenever a space appears. Since the spaces appear most frequently at non-bottleneck stations, parts are dispatched to those stations. However, they cannot get past the bottleneck stations resulting in large queues of waiting parts. The hierarchical control algorithm, on the other hand,uses feedback information to dispatch parts at the current capacity. This results in a smaller inprocess inventory.

A 10% reduction of the demand requirement to 9254 pieces,which is

within the capacity of the implemented interval loading scheme, results in an insignificant drop in total production. However, the production conforms accurately to the requirement. This accuracy is reproduced over several simulation runs each equivalent to 16 hours of production. Thus accurate knowledge of the capacity of the workcenter leads to production plans being met with repeatable accuracy. This is important in situations where the production of several workcenters must be co-ordinated.

Installing a large central storage area results in an increase in the utilization of available time at bottleneck of approximately 5%. The cost, in terms of the decreased inprocess inventory, is substantial.

6.6 Summary

In this chapter, the hierarchical production control algorithm is tested on a three-stage two-part system with 2 machines at each stage and a four-stage four-part system with 4 machines at each stage. The performance of the algorithm conforms to theoretical predictions. It meets production targets accurately and with a small inprocess inventory when the production requirement is within the capacity of the system. Even where the demand requirement is beyond the actual capacity, the algorithm maintains close control of the system, still keeping the inprocess inventory small.

The certainty equivalence (CE) and estimate based (EB) flow controllers are easily executed on-line. The time required to compute the cost estimates for the EB controller are fairly long. However since the estimates are computed off-line and stored, the computational cost is not excessive. The graph of

Figure 6.6 shows that the buffer state is responsive to the production vector u(t) determined by the flow control algorithm.

The four-stage four-part example shows that some improvements to the simple interval loading scheme are necessary. Increasing the utilization of available time beyond the 90% achieved by the interval loading scheme without a substantial increase of inprocess inventory requires a more elaborate sequence control algorithm.

The results of this chapter show that using feedback information on the system has definite advantages. In particular, dispatching pieces at a rate that is within the current capacity of the system reduces the inprocess inventory and the proportion of time that parts spend waiting in internal storage areas. Using downstream buffer state information keeps workcenter production close to the desired demand rates. This is advantageous if the workcenter output goes to downstream manufacturing processes for further processing or assembly because it is then not necessary to maintain large downstream inventory levels in order to smooth out production.

The hierarchical production control algorithm is based on analytical models of the manufacturing system. An alternative approach is to construct heuristic scheduling rules using simulation models [Hutchinson and Hughes, 1977]. Using this approach, good performance can be obtained for a given system configuration. However, if the demand requirements or system parameters change, it may be necessary to adjust the heuristic operating rules for the new configuration. This process can be time consuming since many simulation runs are necessary in order to obtain the best performance for the decision rule.

In Appendix II, we give an example of a heuristic operating rule and compare its performance to that of the EB flow control algorithm. The comparison demonstrates the difference between heuristic and analytical approaches to production planning and control.

CHAPTER VII

7.0 CONCLUSION AND GENERALIZATIONS

7.1 Introduction

This chapter concludes the thesis and suggests areas and open questions that warrant further research. A summary of the thesis is given in Section 7.2. Generalizations possible extensions of the workcenter model and improvements to computational methods are discussed in Section 7.3. The wider problem of production planning and control for an integrated factory with several automated workcenters is described briefly in Section 7.4.

7.2 Summary of Thesis

The problem of planning and controlling production in an automated flexible workcenter with failure prone workstations has been considered. The managers of such a facility are faced with the problem of planning the production of one or more part families. Their objective is satisfy a master production schedule while at the same time, utilizing fully the workstations in the system and minimizing the size of the inprocess inventory.

The automated workcenter is controlled by one or more computers which have the task of scheduling the operations in the system so that production targets set by facility managers are satisfied.

A hierarchical production control algorithm depicted in Figure 2.1, which continuously adjusts the throughput rates for a part family in order to compensate for workstation failures and repairs,is described in Chapter 2. In a flexible workcenter, production is not halted when workstations fail. However, failures reduce the capacity of the system. Furthermore, the effect of a failure on each part differs. It is necessary therefore to adjust the rates at which members of a part family enter the system as machine failures and repairs take place.

The flow control level of the hierarchy models the workcenter as a processing system with a time varying capacity. The instantaneous capacity is the set of possible part mixes that can be produced per unit time by the currently operational workstations. The material flow is modelled by a continuous process. The flow control problem is formulated in Chapter 3 as a stochastic optimal control problem with the throughput rate of the part family as control variable. The optimal control policy is a feedback law which, for each machine state, gives the production rate as a piecewise constant function of the cummulative difference between actual and desired production.

Practical computational schemes which retain the characteristics of the optimal policy are developed in Chapter 4. The estimate based (EB) controller computes off-line, estimates of the optimal cost-to-go function. On-line, the estimates are used to determine the production rate by means of a simple linear program. The certainty equivalence (CE) controller calculates production rates while taking into account the failures and repairs that may occur next in the system.

Whenever a part has a choice of one or more stations for a particular operation, the routing level determines the proportion of parts to be sent to each available station. Feasible routing choices can always be found because the throughput rate set by the flow controller is within the current capacity of the system. Network-of-queues models may be employed to compare different choices when the routing is not unique.

The sequence control level determines the sequence and times at which parts enter the workcenter and controls the routing for individual pieces.

Workcenter managers should impose demand requirements on the system which maximize the utilization of the workstations. Chapter 5 develops the equivalent capacity set Φ of the workcenter which allows the workcenter managers to determine fairly easily the maximum production rate that the system can achieve. A production planning scheme using the equivalent capacity set is suggested.

The hierarchical controller is implemented on a simulated model of a planned electronic circuit board assembly system. The results presented in Chapter 6 show that the algorithm is capable of meeting set production targets provided they are within the capacity of the workcenter. The control system is robust with respect to modelling errors provided that the errors do not result in an overestimation of the productive capacity of the workcenter.

While in the normal course of events the hierarchical controller automatically schedules the operation of the system, compensating for

demand changes and repairs and failures of workstations, parameter estimation and predictive algorithms are necessary to monitor and report to facility personnel the performance of the system. Whenever forecasts of system production indicate that set targets are likely to be missed, because for example, a priori estimates of mean time between failures and repairs are too optimistic, operator intervention can be signalled.

The results of this research show that it is possible to schedule the operations of an automated manufacturing system with unreliable workstations so that downstream requirements are met with a low inprocess inventory. The proposed hierarchical controller therefore realizes an important advantage of flexible automation namely, the ability to produce a family of parts as they are required. This eliminates the need to keep large inventories of parts.

7.3 Workcenter Modelling and Computation

7.3.1 The Failure Model

The workcenter failure model assumes that failure rates are independent of the material flow at the workstations and that the mean time between failures are long compared to part processing times. In this section we suggest ways of relaxing these assumptions.

In a workcenter, we would expect heavily loaded machines to fail more often than lightly loaded ones. This is particularly so if part

related failures such as tool breakages and workpiece jams are a frequent cause of down time.

The failure rate of machine m can be modelled as a function $p_m(w_m)$, where w_m is a vector of flowrates w_{nm}^k of type n parts undergoing operation k at the station. The properties we might expect of $p_m(\cdot)$ are that

$$p_m(0) = p_{mo} > 0 \tag{7.1}$$

and $p_m(w)$ is a strictly monotonically increasing function. This means that an idle machine which is warmed up and in a ready condition but not operating on a part can be expected to fail, perhaps due to electrical, hydraulic or software faults, with a mean time to fail of $1/p_{mo}$. Monotonicity of $p_m(w)$ implies that increasing the flow rate of any part while others are constant results in an increased failure rate.

Assuming memoryless failure processes, the system can still be modelled as a Markov process. The transition rates defined in Chapter 3 become functions $\lambda(w)$ of the flow assignments $w=(w_1,w_2,\ldots,w_M)$. The optimal policy and cost-to-go functions are characterized by a modification of equation (3.20):

$$\min \left\{ \frac{\partial}{\partial t} J_{u^*}(x,i,t) + g(x) + \frac{\partial}{\partial x} J_{u^*}(x,i,t)\,(u(x,i,t)-d(t)) + \sum \lambda_{ij}(w) J_{u^*}(x,j,t) \right\} = 0 \tag{7.2}$$

subject to (3.5), (3.6) and $w \geq 0$, $u \geq 0$.

The optimal control may no longer be at an extreme point of the set $\Omega(i)$. The throughput and routing assignments must be computed simultaneously. However, if $p_m(w)$ is a linear function

$$p_m(w_{nm}^k) = p_{mo} + \sum_n \sum_k p_{nm}^k w_{nm}^k \tag{7.3}$$

where p_{nm}^k are positive constants, the piecewise constant characteristics of the optimal policy described in Chapter 3 is unchanged.

The effect of different types of operations on the reliability of the workstations should be empirically determined. The linear model (7.3) is probably a good initial hypothesis. Extensive data on an operational workcenter is required in order to establish estimates for the parameters p_{nm}^k. This may prove to be difficult especially where production changes frequently between different part families.

Simulation results show that the bounding technique produces good operating policies. We can obtain bounds for systems in which λ_{ij} is a function of w by using transition rates $\overline{\lambda}_{ij}$ and $\underline{\lambda}_{ij}$. They are chosen such that for all w_{nm}^k,

$$\underline{\lambda}_{ij} \leq \lambda_{ij}(w) \leq \overline{\lambda}_{ij} \tag{7.4}$$

The estimates of the cost-to-go functions can be used as before in real time to calculate the flow rates through the system.

Experience with the simulation model shows that using the upper bound estimates gives good hedging behavior when failure rates do not

depend on the material flow through the workstations. With the failure model of equation (7.3), the failure rate of bottleneck stations is close to the upper limit in (7.4). Therefore,a policy that uses upper bounds is likely to compensate sufficiently for bottleneck station failures. However, if non-bottleneck stations have failure rates that are less than $\bar{\lambda}_{ij}$, the policy may be too cautious. A simulation test is needed to show whether or not this results in a degradation of performance.

The calculation of the equivalent production constraint sets Φ and $\Phi_T(x,\alpha)$ can be made using the upper bound values $\bar{\lambda}_{ij}$. Since bottleneck machines are the most utilized, their failure rates are close to the maximum. The estimated capacity is therefore likely to be fairly good. A test of this conjecture can be made when actual data becomes available.

Dependence of failure rates on material flow through the workstations makes the parameter estimation discussed in Section 2.5 especially important and difficult. Whenever production is started on a new part family, it will take some time before accurate assessments of mean time between failures are made. The managers of the facility may then have to review production plans frequently if the long term objectives are to be met.

In our model, we assume that mean time between failures (MTBF) and repairs (MTTR) are long compared with the time required to produce

a single part. In metal cutting, for example, the time required to complete a part may be of a length comparable to the MTBF or MTTR. This is especially true if the parts are large [Hatschek, 1979]. When operation times are that long, a piece going through a workcenter has a relatively high probability of being affected by a workstation failure. In this case, down-times can be modelled us random disturbances on the operation times.

Let τ^k_{nm} be the average time taken by a type n piece to complete operation k at station m. The average time τ^k_{nm} includes the effect of down-times. The flow control level of the hierarchical controller chooses throughput rates $u^n(t) \geq 0$ for part n and operation splits $w^k_{nm} \geq 0$ that satisfy

$$u^n(t) = \sum_m w^k_{nm} \qquad \forall k \qquad (7.5)$$

$$\sum_{n,m} w^k_{nm} \tau^k_{nm} \leq 1 \qquad \forall m \qquad (7.6)$$

with the objective of minimizing the performance index(3.8).

The control vector $u(t)$ is thus chosen in a constraint set defined by (7.5)-(7.6) which is based on the average availability of the workstations. The instances at which the flow control level of the hierarchy changes u(t) therefore do not necessarily correspond to times at which failures or repairs occur. This is because the buffer

state changes by small amounts between machine state changes due to the long operation times.

The operation of the sequence control level of the control hierarchy is more complicated than in the situation where the MTBF and MTTR are long compared with processing times. The objective is still to maintain the chosen flow rates $u(t)$. However, the sequence controller must now take into account the fact that failures result in random processing times for the parts at the workstations. Scheduling rules are needed for the sequence control level which maintain the chosen throughput rates while keeping the number of workpieces inside the workcenter at a minimum. The scheduling rules would have to adapt to new estimates of machine failure and repair rates as they become available.

Predictions of workstation utilization provided by the flow control level may also allow the sequence controller to vary on-line process plans for individual parts. For example, at a lightly loaded machine, feed rates could be reduced without reducing the overall throughput thereby reducing tool wear.

7.3.2 Inspection and Part Rejection

One feature not included in our model is inspection and part rejection. A typical workcenter incorporates inspection stations where parts may be rejected or sent for rework if they do not meet design specification. There may also be stations specially set aside for

rework. Part rejection adds extra random disturbances into the system. Rejection and rework rates should be continuously monitored because they affect the capacity of the workcenter.

As far as the flow control level of the hierarchy is concerned, part rejection and rework changes the capacity of the system and alters the dynamics of the buffer state. Consider for example, the workcenter of Figure 7.1 which includes inspection and rework stations. All parts leaving the workstations are inspected. Let β be the proportion of parts that are inspected rejected and sent for scrap. Define η to be the proportion of parts that are sent for rework. Assuming that a part can only be sent for rework once, the buffer state dynamics can be modelled by

$$\frac{dx(t)}{dt} = [1-\beta(1+\eta)]u(t) \tag{7.7}$$

$$u(t) \in \Omega(\alpha(t)) \tag{7.8}$$

With the restrictions $(\beta+\eta)<<1$, $\eta \geq 0$ and $\beta \geq 0$. There may also be capacity limitations on the rework and inspection stations. The scalars η and β may be replaced by diagonal matrices if the rejection and rework rates differ among the parts.

Inspection and rejection after each operation within the workcenter means that at the routing algorithm level, the order in which parts visit the workstations may have to be taken into account in solving the routing problem. Additional constraints may result since it may be

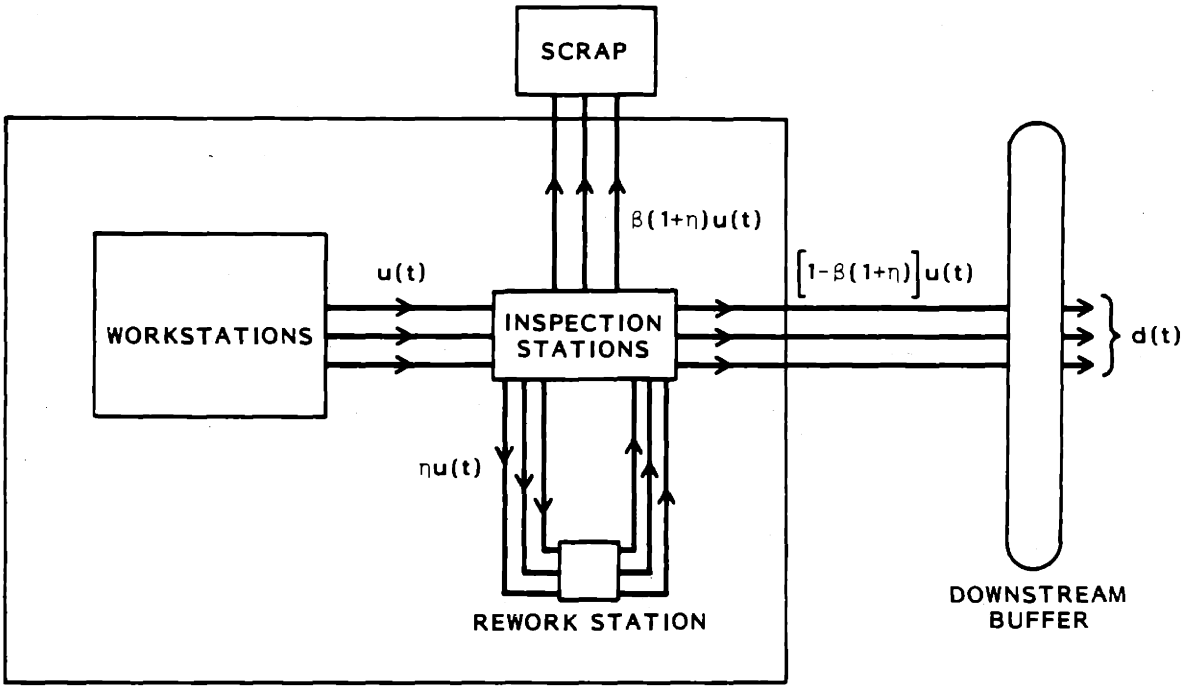

Figure 7.1 A Workcenter With Inspection and Rework Stations.

preferable to send parts to stations with the lowest rejection rate whenever there is a choice. In workcenters where the nature of the operations is such that there are few precedence constraints, operations that are relatively cheap to perform or those that have higher rejection rates should perhaps be done first. This is because it is preferable to scrap a piece that has had a relatively small amount of work put into it rather than one that is almost finished or has had the most costly operations in terms of time and material.

In Section 3.1, it is mentioned that scheduled maintenance can be modelled as a set of parts which need operations at the workstation with a duration equal to the required maintenance period. The production rate is determined by the frequency of maintenance. The scheduling of maintenance could then be done by the hierarchical controller and could also be related to the amount of work performed at the workstations. Scheduled down-time should be incorporated into production planning and control algorithms, so that downstream buffer levels are kept at levels that adequately hedge against planned and unplanned downtime.

7.3.3 Reduction of Computational Effort at the Flow Control Level

Estimates of the optimal cost-to-go functions are determined in Section 4.3.2 by fitting hypercubes to the control constraint sets and then applying the successive approximations algorithm to the resulting

decoupled problem. Simulation results in Chapter 6 indicate that the shape of the level surfaces of the estimates are more important than their values because they determine the hedging points and the locations of the boundaries of the control regions. It would be useful to explore ways of exploiting these facts in order to speed up the calculation of the cost estimates. A reduction in the off-line computational burden would not only increase the size of the system in which the estimate based (EB) sub-optimal controller can be applied, it would also make it easier to recompute policies when new parameter estimates become available.

The off-line computational requirements for the EB controller can be considerably reduced by taking advantage of the fact that a large number of the machine states are either very unlikely or have a control constraint set which consists of a single point at the origin. In the 4-stage 4-part example, the control vector $u(t)$ is zero whenever all four machines are failed at the same stage. The number of machine states for which estimates of the optimal cost-to-go functions are needed is therefore reduced from 625 to 256. A systematic method of reducing the state space would result in savings in computational and storage requirements.

The certainty equivalence (CE) approach to sub-optimal control presented in Section 4.4, involves solving on-line mathematical

programming problems which may become fairly large in practice. These problems have a special structure because the constraints (4.53) represent dynamic variables and (4.54)-(4.55) are decoupled, because they represent distinct control vectors. By exploiting the structure of the problem, it may be possible to develop efficient computational techniques. The CE approach could then be applied to systems with a larger number of parts and machines.

7.3.4 Improvement to the Sequence Control Level

Simulation results for the 4-part 4-stage example in Section 6.5 show that the capacity of the system is less than that predicted in Chapter 5 because the sequence controller implemented in the model does not achieve 100% utilization of available time at bottleneck workstations. The implemented controller is extremely simple. It does not vary actual loading times within the window but instead loads parts as soon as possible. Likewise, when an operation is completed, the sequence controller moves the part immediately to the first available workstation.

Improvements to the lowest level of control can be realized if all routing decisions affecting individual pieces are made by the sequence controller. The decisions would be made with the objective of maintaining the flow rates chosen by the flow and routing levels of the control hierarchy. A part entering the system should not have the

sequence of workstation visits predetermined. Instead, the sequence controller should resolve each operation or routing choice as the part progresses through the system using the flow rates and operational splits as set points on which to base decisions. The sequence controller should also have the ability to look ahead and optimize the times within the calculated windows at which parts enter the system and commence operations.

Any improvement in the effectiveness of the sequence controller would yield gains in productivity because increasing the utilization of bottleneck stations results in a higher throughput without an attendant increase in the in-process inventory.

7.4 The Overall Production Planning Problem

In this research, we have shown that a controller for an automated workcenter can accurately track specified demand requirements. A factory consists of an interconnection of several flexible and non-flexible workcenters each one locally controlled, and all co-ordinated through the master production plan. The effect of applying local feedback control on the ability of the factory to meet production requirements and the size of the inventory of parts and components remains to be investigated.

Traditionally, workcenters are decoupled by large amounts of storage. It is not unusual for example, to have several shifts worth

of components upstream of an assembly line [Gershwin et al., 1980]. Certain Japanese manufacturers by carefully coordinating production, are able to have components arrive at workcenters as they are required. The need for large inprocess inventories is thereby eliminated realizing considerable savings in production costs [Lohr, 1982].

The coordination of a number of connected flexible workcenters requires careful study. We can assume that each workcenter is capable of tracking demand rates accurately if all raw material is available. The coordination problem is thus one of setting production targets for each workcenter and controlling storage between the centers.

The hierarchical controller's feedback policy sets throughput rates as a function of the machine state and, $x(t)$, the difference between actual and desired production. If the downstream manufacturing process is also unreliable, the actual contents of the downstream storage do not affect the upstream production policy. This is because the rate at which material is removed from the storage by the downstream process is different from the desired rate $d(t)$. There is therefore the possibility that the storage will fill up and block the upstream workcenter or it will empty and starve the downstream manufacturing process.

On the other hand, if the control policy is a function of actual downstream buffer contents, the workcenters production will track the downstream processe's production rate. If the downstream workcenter

performs poorly because of higher than average failures, the low production rate is propagated upstream resulting in a poor overall factory performance.

The local feedback policy for each workcenter should therefore have the dual objective of producing the desired quantity of material and at the same time minimizing the possibility of starving the downstream workcenters. The material flow process may be complex because a workcenter can supply more than one downstream manufacturing system which may in turn remove parts from the storage in batches with a specified number of parts of each type in a batch.

The workcenter model of Chapter 3 assumes that the demand for a part family is known and deterministic. In an interconnected system, the workcenter is called upon to supply downstream processes which may have unreliable elements. The downstream demand may therefore be random. The flow control policies characterized in Chapter 3 should be extended to cover the situation in which the downstream demand $d(t)$ is stochastic. Random demands may also be created by orders for parts not in the master production plan. In this case, the quantity and processing requirements for these parts may not be known in advance at the workcenter level. Techniques for handling emergency and "rush" orders for parts in a way that minimizes disruption to planned production should be developed.

Lasserre [1979] has studied medium term planning over a horizon of several weeks for a semiconductor production unit. His system consists

of several workcenters arranged in series with buffers decoupling adjacent centers. By exploiting the serial structure of the manufacturing system and the fact that different kinds of parts have the same processing times in the workcenters, efficient algorithms are developed for calculating rolling horizon production plans for each workcenter.

The problem of coordinating several arbitrarily connected flexible workcenters producing part families with different processing times at the workstations should be addressed. Suitable models are needed for the material flow processes within such an integrated system. Efficient planning tools should be developed with which the managers of the factory can schedule production requirements on each workcenter so that the demand for final products is satisfied with a minimum inventory of inprocess parts.

The information flow process is just as important as material flow. A vast amount of data is collected by sensors on the factory floor and from personnel at all levels of management. The decisions that are made in operating the factory require information about the status of the system. This implies that the design of the system data base should go hand in hand with the development of the planning and control system. The objective is to provide each decision maker with timely and relevant information on which to base its actions.

The trend evident in some of the flexible systems described in Chapter 1 is to install a hierarchy of several computers to manage the operations of various sections of the factory and within the workcenters. This is so that the control system itself can be made more reliable through the redundancy that is introduced [Bruner et al., 1981]. The data base therefore need not be centralized. Each controller can maintain detailed local information while passing aggregated data to neighboring controllers and to the coordinating levels of the production planning and control system.

REFERENCES

[1] Aggarwal, S. C. "A Review of Current Inventory Theory and its Application," Int. J. of Proc. Res., Vol. 12, No. 4, 1974, pp. 443-482.

[2] Athans, M., Cook, N. H., Gershwin, S. B., Horev, Y., Kanellakis, P. C., Kimemia, J., Schick, I. C., and Ward, J. E. "Complex Materials Handling and Assembly Systems-Second Interim Process Report," M.I.T. Laboratory for Information and Decision System Report ESL-FR-771.

[3] Baskett, F., Chandy, M., Muntz, R., and Palacios, F., "Open Closed and Mixed Networks of Queues with Different Classes of Customers," Journal of the A.C.M., Vol., 22, No., 2, April 1975.

[4] Beecher, R. C., and Dewar, R., "Robot Trends at General Motors," American Machinist, 1979.

[5] Bertsekas, D. P. Dynamic Programming and Stochastic Control, Academic Press, 1976.

[6] Blackburn, J. D., and Millen, R. A. "The Impact of a Rolling Schedule in a Multi-Level MRP Systems,". ORSA/TIMS National Meeting, Colorado Springs, November, 1981.

[7] Borzcik, P. S., "Flexible Manufacturing Systems," Technology of Machine Tools Vol., 2: Machine Tools Systems Management and Utilization, Lawrence Livermore Laboratory, October 1980, pp. 62-74.

[8] Bradshaw, A. and Erol, Y. "Control Policies for Production-Inventory Problems With Bounded Input," Int. Journal of Systems Science, Vol., 11, no. 8, 1980.

[9] Bruner, R. H., Holden, E. J., Luber, J. C., Mozer, D. T. and Ning-Gau Wu, "Automated Semiconductor Line Speeds Custom Chip Production," Electronics, January 27, 1981.

[10] Buzacott, J. A. "Automatic Transfer Lines with Buffer Stocks" International Journal of Production Research, Vol. 5, No. 2, pp. 183-200, 1967.

[11] Buzacott, J. A., ""Optimal" Operating Rules for Automated Manufacturing Systems" IEEE Transactions on Automatic Control, Vol., AC-27, No. 1, Feb. 1982.

[12] Buzacott, J. A., and Yao, D. "Flexible Manufacturing Systems: A Review of Models," University of Toronto, Dept. of Industrial Engineering Working Paper #82-007 March 1982 (Also to appear at TIMS/ORSA Detroit Meeting, Apr. 82).

[13] CAM-I, "Factory Management Project" Computer Aided Manufacturing International Report No. PR-81-ASPP-01.6, 1982

[14] Chong, C. and Athans, M. "Hierarchical Decomposition for a Stochastic Optimization Problem," Proc. of the 7th Annual Princeton Conference on Information Sciences and Systems,Princeton, New Jersey, 1973.

[15] Cook, N. H. "Computer Managed Parts Manufacture," Scientific American, Vol., 232, February 1975, pp. 23-29.

[16] Cruz, Jr., J. B. "Stackelberg Strategies for Multilevel Systems," In Directions in Large-Scale Systems, Plenum Press, New York., pp. 139-149, 1975.

[17] Dallas, D. "The Robot Enters the Systems," Manufacturing Engineering, February 1979.

[18] Deuernmeyer, B. L. and Pierskalla, W. P. "A By-Product Production System with an Alternative," Management Science, Vol. 24, No. 13 September 1978.

[19] Economist "Car Automation: Robots Change the Rules," April 19th, 1980. pp. 93-94.

[20] Evershein, W. and Westkamper, E. "Development of Automated Manufacturing Systems," Fifth North American Metalworking Research Conference, University of Massachusetts, Amherst, May 1977.

[21] Gershwin, S. B., Ward,J. and Athans, M. "Complex Materials Handling and Assembly Systems Final Report," Vol. 1, M.I.T., Laboratory for Information and Decision Systems, Report No. ESL-FR-834-1, November 1980.

[22] Halevi, G. The Role of Computers in Manufacturing Processes, John Wiley and Sons, 1980.

[23] Hahne, E. "Dynamic Routing in an Unreliable Manufacturing Network with Limited Resources," M.I.T. Laboratory for Information and Decision Systems, Report No. LIDS-TH-1063, 1981.

[24] Hatschek, R. L. "Dual Head Changers for Short Runs," American Machinist, March 1979.

[25] Hartwig, A. "Small Machining Centers Bring High Technology to the Job Shops," Manufacturing Engineering, February 1982.

[26] Hildebrandt, R. R. "Scheduling Flexible Machining Systems When Machines are Prone to Failure," PhD Thesis, M.I.T., Dept. of Aeronautics and Astronautics, May 1980.

[27] Hitomi, K. Manufacturing Systems Engineering, Taylor and Francis,London, 1979.

[28] Hitomi, K., Nakamura, N. "Optimal Production Planning for Multiproduct, Multistage Production Systems," Int. Journal Prod. Res., Vol. 4, No., 2, 1976, pp. 199-213.

[29] Hitz, K. "Scheduling of Flexible Flow Shops - 11, "M.I.T. Laboratory for Information and Decision Systems, Report No. LIDS-R-1049, October 1980.

[30] Ho, Y. C. and Cassandras, C. "Computing Co-State Variables for Discrete Event Systems," IEEE Conference in Decision and Control, San Diego, CA, December 1980.

[31] Hughes, J. J., Hutchinson, G. K., and Gross, K.E. "Flexible Manufacturing Systems for Improved Mid-Volume Productivity," Understanding Manufacturing Systems, Vol., 1, Kerney and Trecker Co., 1978.

[32] Hutchinson, G. K., "The Control of Flexible Manufacturing Systems: Required Information Structures," IFAC Symposium on Information-Control Problems in Manufacturing Technology, Japan, October 1977.

[33] Hutchinson, G. K., and Hughes, J. J. "A Generalized Model of Flexible Manufacturing Systems," Multi-Station Digitally Controlled Manufacturing Systems Workshop, University of Wisconsin, Milwaukee, January 1977.

[34] Isobe, T. "Automatic Control in the Iron and Steel Industry," Automatica, Vol., 6, No., 1, January 1970.

[35] Kanellakis, P. "Algorithms For a Scheduling Application of the Asymmetric Travelling Salesman Problem," M.I.T. Laboratory for Information and Decision Systems Report No. ESL-R-834-5, 1978.

[36] Karmarkar, U.S. "Convex|Stochastic Programming and Multilocation Inventory Problems," Naval Research Log. Quarterly, March 1979.

[37] Kimemia, J, and Gershwin, S. B. "Multicommodity Network Flow Optimization of a Flexible Manufacturing System," MI.T. Laboratory for Information and Decision Systems Report ESL-FR-834-2.

[38] Kleindorfer, P. R., Kriebel, C. H. Thomson, A. L. and Kleindorfer, A.B. "Discrete Optimal Control of Production Plans," Management Science, Vol. 22 No. 3, November 1975.

[39] Kleinrock, L. Queueing Systems, Volume I: Theory, John Wiley and Sons, New York, 1975.

[40] Kushner, H. J. "Optimality Conditions for the Average Cost per Unit Time Problem With a Diffusion Model," SIAM J on Control and Optimization, Vol. 16, No. 12, March 1978.

[41] Kushner, H. J. Introduction to Stochastic Control, Holt, Rinehart and Winsten,Inc. New york, 1971.

[42] Lassere, J. B. and Roubellat, F. "Middle Term Planning for a Semicondutors Production Unit," 5th International Conference on Production Research, Amsterdam, Holland, August 1979.

[43] Lerner, E. J. "Computer Aided Manufacturing," IEEE Spetrum, Vol. 8, No. 11, November 1982.

[44] Lewald, R. "Flexible System Makes Aircraft Parts," American Machinist, March 1981, pp. 107-110.

[45] Lohr, S. "The Company that Stopped Detroit," New York Sunday Times, April 21, 1982.

[46] Manufacturing Engineering "Putting Robotics To Work at Massey-Ferguson," June 1979, pp. 80-81.

[47] Mesarovic, M. D., Macko, D. and Takahara, Y. Theory of Hierarchical Multi-level Systems, Academic Press, 1970.

[48] Nof, S. Y., Barash, M. M. and Solberg, J. "Operational Control of Item Flow in Versatile Manufacturing Systems," Int. j. of Production Research, Vol, 17, No. 5, 1979, pp. 479-489.

[49] Nof, S. Y. and Solberg, J. J. "Control Considerations in Layout Configurations for Manufacturing Systems," J on Dynamic Systems and Control, Trans of the ASME Vol., 102, No., 3, September 1980.

[50] Olsder, G. J. and Suri, R. "Time Optimal Control of Parts Routing in a Manufacturing System with Failure-Prone Machines," IEEE Conference in Decision and Control, San Diego, CA, 1979.

[51] Orlicky, J. Material Requirements Planning, McGraw-Hill, 1975.

[52] Pressman, R. S., and William, J. E. Numerical Control and Computer Aided Manufacturing, John Wiley and Sons, New York, 1977.

[53] Rishel, R. "Dynamic Programming and Minimum Principles for Systems with Jump Markov Disturbances," SIAM Journal on Control, Vol., 13, No. 2, February 1975.

[54] Schaffer, A. "P C Supervises CDCs, Conveyer, Robots in High-Volume Production Line," American Machinist, October, 1978.

[55] Secco-Suardo, G. "Workload Optimization of an FMS Modeled as Closed Networks of Queues," Annals of the CIRP, Vol. 28, No. 1, pp. 381-383, 1979.

[56] Shanthikumar, J. G. and Sargent, R. G. "A Hybrid Simulation|Analytic Model of a Computerized Manufacturing Systems," Syracuse University Dept. of Industrial Engineering and Operations Research Working Paper #80-017, December 1980.

[57] Skole, R. "Unmanned Machining at Work," American Machinist, June 1979.

[58] Stecke, K. E. "Experimental Investigation of the Scheduling Problems of a Particular Computerized Manufacturing Systems," Proc. of Conference on Optimal Planning of Computerized Manufacturing Systems, Purdue University, November 1977.

[59] Sullivan, W. "The Factory of the Future" The New York Times, Jan. 10, 1979.

[59] Tsitsiklis, J. N. "Characterization of Optimal Policies in a Dynamic Routing Problem," M.I.T. Laboratory for Information and Decision Systems, Report No. LIDS-R-1178, February 1982.

[60] Veinott, Jr., A. F. "Optimal Policy for a Multi-Product Dynamic Non-stationary Inventory Problem," Management Science, Vol. 12, No. 3, November 1965.

[61] Wick, C. "Machining Center Update," Manufacturing Engineering, Sept., 1979.

[63] Winship, J. T. "Update on Industrial Robots," American Machinist, Jan., 1979.

[64] Zangwill, W. I. "A Deterministic Multiproduct Multi-Facility Production and Inventory Model," Operations Research, Vol. 14, No. 3, 1966, pp. 486-507.

APPENDIX A

A PROPERTIES OF OPTIMAL POLICIES

In this appendix, we look at some of the technical points of the flow control model presented in Chapter III. For u in the class of admissible controls, let $\theta_{i\ell} \subset R^N \times (0,T)$ be a region within which u(x,i,t) is a continuous function of (x,t). Given an initial condition $(x,t) \in \theta_{i\ell}$, the corresponding deterministic trajectory $x_u^i(s;x,t)$ defined by (3.13) passes through a sequence of regions $\{\theta_{i\ell}\}$ $\ell = \ell_0,\ldots,\ell_v$ until the final state and time are reached at $(x_u^i(T;x,t),T)$.

The regions $\theta_{i\ell}$ can be further divided into sets C_{ij} for j in an index set J. Each set C_{ij} is an open subset of a region $\theta_{i\ell}$ and for all $(x,t) \in C_{ij}$, the trajectories $x_u^i(s;x,t)$ pass through the same sequence of sets C_{ik}, $k \in J$. Since C_{ij} is contained in some region $\theta_{i\ell}$, u(x,i,t) is continuous for all (x,t) in C_{ij}.

From the definition of the sets C_{ij}, $j \in J$, for (x,t) in C_{ij_0}, $x_u^i(s;x,t)$ passes through a sequence of sets $\{C_{ij}\}_{j=(j_0,j_1,\ldots,j_q)}$ until it terminates at $(x_u^i(T;x,t),T) \in C_{ij_q}$. The boundary B_{j_k} between two adjacent sets C_{ij_k} and $C_{ij_{k+1}}$ is defined as

$$B_{j_k} = Cl(C_{ij_k}) \cap Cl(C_{ij_{k+1}}) \tag{A.1}$$

where Cl(A) is the closure of the set A. If B_{j_k} is smooth in the sense that there is a differentiable function $f_{ij_k}(x,t)$ such that for every $(x,t) \in B_{j_k}$,

$$f_{ij_k}(x,t) = 0 \tag{A.2}$$

then it follows that $x_u^i(s;x,t)$ is a continuous function for all (x,t) in $\bigcup_{k=0}^{q} C_{ij_k}$ [Rishel, 1975]. If in addition, $u(x,i,t)$ is differentiable for all $(x,t) \in C_{ij}$, then $x_u^i(s;x,t)$ is differentiable in the same sets .

Consider the stochastic optimal control problem (3.8) and the cost-to-go function $J_u(x,i,t)$ defined by (3.12). Equation (3.14) follows from (3.12) [Rishel, 1975] if we integrate along the deterministic trajectory $x_u^i(s;x,t)$ and make use of the fact that the probability that there is no transition from machine state i in the interval (t,s) is given by $e^{\lambda_{ii}(s-t)}$. Since $x_u^i(s;x,t)$ is differentiable within the sets C_{ij}, (3.14) can be differentiated to yield (3.15).

The relationship (3.16) for an optimal policy follows because if there is a policy $\tilde{u}$ for which

$$\frac{\partial}{\partial x} J_{u^*}(x,i,t) \; [\tilde{u}(x,i,t)-d(t)] < \frac{\partial}{\partial x} J_{u^*}(x,i,t)[u^*(x,i,t) - d(t)] \tag{A.3}$$

then over a suitably small interval of time δt, define a new policy

$$u(x,i,s) = \begin{cases} \tilde{u}(x,i,s) & t \leq s \leq t + \delta t \\ u^*(x,i,s) & \text{Otherwise} \end{cases}$$

It can be shown that $J_u(x,i,t) \leq J_{u^*}(x,i,t)$ [Rishel, 1975] hence u^* could not have been an optimal policy. In fact, it can be shown that (3.16) is both necessary and sufficient.

For an optimal policy, we can identify $\theta_{i\ell}$ with the regions in which $u^*(x,i,t)$ takes values at an extreme point of $\Omega(i)$.

A.1 Behavior of the Buffer State Trajectory in the Neighborhood of a Region Boundary

Consider a trajectory $x^i_{u^*}(s;x,t)$ resulting from an optimal control policy. Let the initial condition (x,t) be in the interior of $C_{ij_k} \subseteq \theta_{i\ell}$. From the definition of C_{ij} and $\theta_{i\ell}$, $x^i_{u^*}(s;x,t)$ travels through C_{ij_k} under the action of a constant control $u^*_{ij_k}$ until it reaches the boundary of C_{ij_k} and $C_{ij_{k+1}}$. From (A.1) and (A.2) the closure of C_{ij_k} is a subset of

$$B_{j_k} = \{(x,t) \mid f_{ij_k}(x,t) = 0\} \tag{A.4}$$

where f_{ij_k} is a differentiable function. The optimal controls in C_{ij_k} and $C_{ij_{k+1}}$ are $u^*_{ij_k}$ and $u^*_{ij_{k+1}}$ respectively and without loss of generality, it can be assumed that $f_{ij_k}(x,t) < 0$ for $(x,t) \in C_{ij_k}$ in the neighborhood of the closure of C_{ij_k}. There are two cases to consider at a point (x,t) on the boundary:

$$(u^*_{ij_{k+1}} - d(t))' \frac{\partial f_{ij}(x,t)}{\partial x} + \frac{\partial f_{ij}(x,t)}{\partial t} > 0 \tag{A.5}$$

illustrated in Figure A.1, and

$$(u^*_{ij_{k+1}} - d(t))' \frac{\partial f_{ij_k}(x,t)}{\partial x} + \frac{\partial f}{\partial t} \leq 0 \tag{A.6}$$

which is shown in Figure A.2.

In the first case, the trajectory $x^i_{u^*}(s;x,t)$ crosses the boundary and proceeds in the set $C_{ij_{k+1}}$ under the control $u^*_{ij_{k+1}}$. In the second

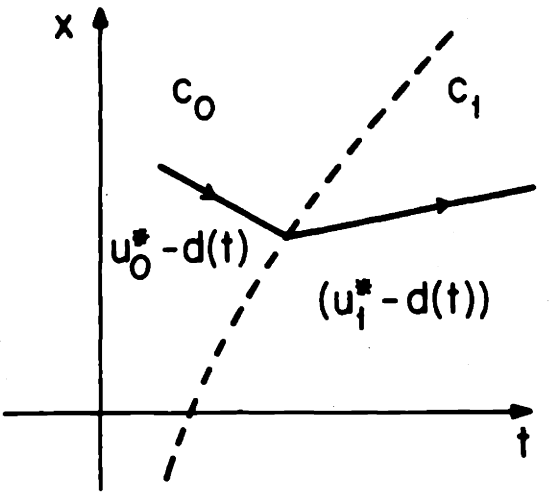

Figure A.1 Trajectory (x(s;x,t),s) crossing from C_0 to C_1

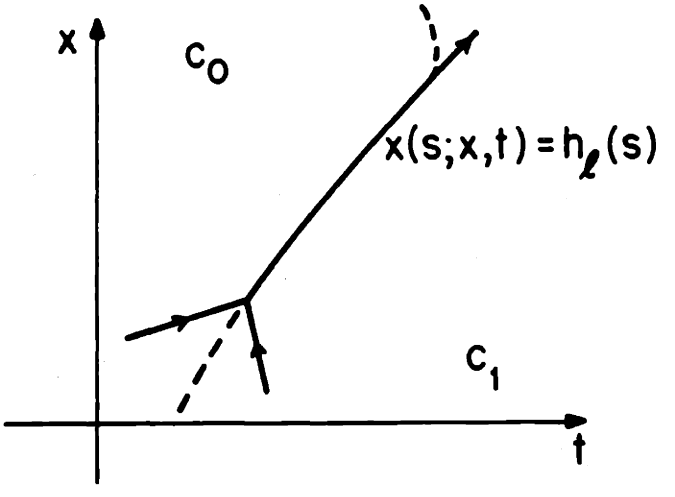

Figure A.2 Trajectory (x(s;x,t)s) Travelling Along Boundary

case, the trajectory travels along the boundary under a control $u^*(x,i,t)$ which is a convex combination of $u^*_{ij_k}$ and $u^*_{ij_{k+1}}$. By the implicit function theorem and (A.4), there is a differentiable function $h_{ij_k}(t)$ such that

$$f_{ij_k}(h_{ij_k}(t),t) = 0 \qquad \forall\ (h_{ij_k}(t),t) \in Cl(C_{ij_k}) \tag{A.7}$$

along the boundary, it follows that

$$x^i_{u^*}(s;x,t) = h_{ij_k}(s). \tag{A.8}$$

The optimal control on the boundary is then given by

$$u^*(x,i,t) = \frac{d}{dt} h_{ij_k}(t) - d(t). \tag{A.9}$$

A boundary where condition (A.6) holds may be considered an additional region associated with the time varying control (A.9). This is similar to singular behavior in deterministic control problems.

A.2 Continuity of the Optimal Cost-to-go Function

In Appendix A.1 it is stated that the cost-to-go function is continuous within the set C_{ij} in which the control is continuous. Now we show that the optimal cost-to-go $J^i_{u^*}(x,t)$ is a continuous function of (x,t) everywhere.

Consider neighboring sets C_{ij_0} and C_{ij_1} depicted in Figure A.3 with optimal controls $u^*_{ij_0}$ and $u^*_{ij_1}$ respectively such that

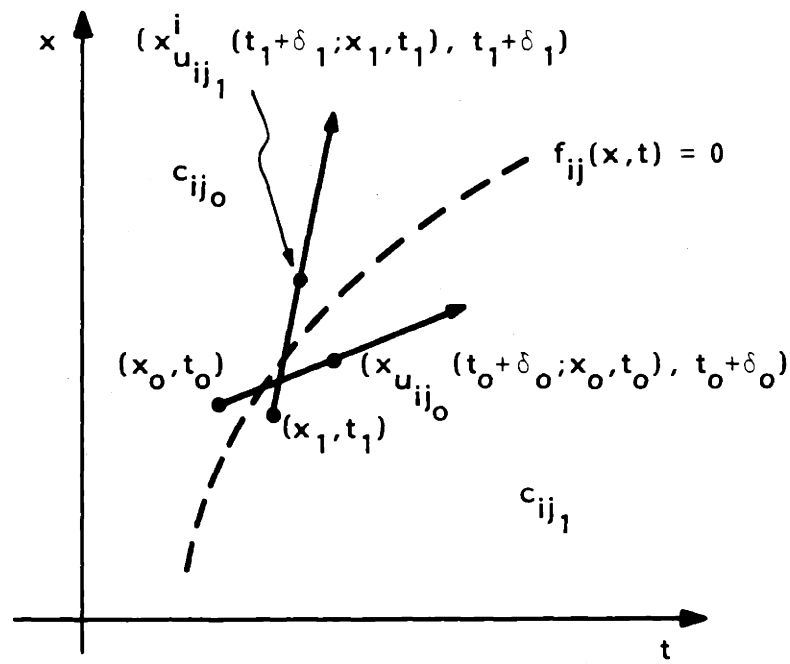

Figure 3.5 The Regions C_{ij_0} and C_{ij_1} Showing the Trajectories due to $u^*_{ij_0}$ and $u^*_{ij_1}$

$$[u^*_{ij_0} - d(t)]' \frac{\partial f_{ij_0}}{\partial x} + \frac{\partial f_{ij_0}}{\partial t} < 0 \qquad \text{(A.10)}$$

$$[u^*_{ij_1} - d(t)]' \frac{\partial f_{ij_0}}{\partial x} + \frac{\partial f_{ij_0}}{\partial t} > 0 \qquad \text{(A.11)}$$

Let $(x_0,t_0) \in C_{ij_0}$ and $(x_1,t_1) \in C_{ij_1}$ be two points in the neighborhood of the boundary such that for a small constant ε,

$$|(x_0,t_0)-(x_1,t_1)| \leq \varepsilon \qquad \text{(A.12)}$$

Starting at (x_0,t_0), apply the control $u^*_{ij_1}$. The resulting cost is given by

$$J_1(x_0,i,t_0) = \int_{t_0}^{t_0+\delta_0} g(x^i(s;x_0,t_0)) + \sum \lambda_{ij} J_1(x^i(s;x_0,t_0),j,s)ds + J_{u^*}(x^i(t_0+\delta_0;x_0,t_0),i,t_0) \qquad \text{(A.13)}$$

where $\delta_0 > 0$ is such that $(x^i(t_0+\delta_0;x_0,t_0),t_0+\delta_0) \in C_{ij_1}$. Inequality (A.10) and (A.11) imply that such a contant exists. Since $J_1(x_0,i,t_0)$ is bounded, we have for a constant $Q > 0$,

$$J_{u^*}(x_0,i,t_0) \leq J_1(x_0,i,t_0) \leq \delta_0 Q + J_u^*(x^i(t_0+\delta_0;x_0,t_0),i,t_0+\delta_0) \qquad \text{(A.14)}$$

A discontinuity at the boundary would imply that

$$J_{u^*}(x_0,i,t_0) < J_{u^*}(x^i(t_0+\delta_0;x_0,t_0),i,t_0+\delta_0) \tag{A.15}$$

Similarly, applying the control $u^*_{ij_0}$ at the point (x_1,t_1) yields for a small constant δ_1 and the resulting cost $J_2(x_1,i,t_1)$

$$J_{u^*}(x_1,i,t_1) \leq J_2(x_1,i,t_1) \leq \delta_1 Q + J_{u^*}(x^i(t_1+\delta_1;x,t_1),i,t_1+\delta_1) \tag{A.16}$$

a discontinuity at boundary implies

$$J_{u^*}(x_1,i,t_1) < J_{u^*}(x^i(t_1+\delta_1;x_1,t_1),i,t_1+\delta_1) \tag{A.17}$$

which contradicts (3.32). We conclude therefore that $J_{u^*}(x,i,t)$ is continuous across the boundary of C_{ij_0} and C_{ij_1}.

I.3 Convexity of the Optimal Cost-to-function

Convexity of $J_{u^*}(x,i,t)$ results from the convexity of $g(x)$ and the linear dynamics of the buffer state trajectory [Tsitsiklis, 1982]. Let u_1 and u_2 be two admissible policies and x_1 and x_2 two points in R^N such that for $\varepsilon > 0$

$$J_{u_1}(x_1,i,t) \leq J_{u^*}(x_1,i,t) + \varepsilon \tag{A.18}$$

$$J_{u_2}(x_2,i,t) \leq J_{u^*}(x_2,i,t) + \varepsilon \tag{A.19}$$

Define a new policy u as

$$u(x,i,t) = cu_1(x_2,i,t) + cu_2(x_2,i,t) \tag{A.20}$$

where x is given by

$$x = cx_1+(1-c)x_2 \tag{A.21}$$

and $c \in [0,1]$. Because the control constraint set is convex, u is feasible. It then follows for buffer state trajectories due to u, u_1 and u_2 that

$$x_u(s;x,t) = c\, x_{u_1}(s;x,t) + (1-c)\, x_{u_2}(s;x,t) \tag{A.22}$$

Consequently by convexity of g(x),

$$J_u(x,i,t) = E\left\{\int_t^T g(x_u(s))ds\right\} \leq E\left\{\int_t^T cg(x_{u_1}(s)) + (1-c)g(x_{u_2}(s))\, ds\right\}$$

$$= c\, J_{u_1}(x_1,i,t) + (1-c)J_{u_2}(x_2,i,t) \tag{A.23}$$

Using (A.18) and (A.19) and the optimality of u*, it follows that

$$J_{u^*}(x,i,t) \leq J_u(x,i,t) \leq c\, J_{u^*}(x_1,i,t) + (1-c)J_{u^*}(x_2,i,t) + \varepsilon \tag{A.24}$$

The constant ε is arbitrary thus we can let it go to zero, in which case we obtain the desired result

$$J_{u^*}(cx_1+(1-c)\, x_2,i,t) \leq cJ_{u^*}(x_1,i,t) + (1-c)J_{u^*}(x_2,i,t) \tag{A.25}$$

APPENDIX B

B.0 AN EXAMPLE OF A HEURISTIC DECISION RULE

In this Appendix, we give an example of a heuristic operating rule for the assembly system described in Section 6.2. We compare its performance with that of the estimate based (EB) flow control algorithm by repeating the tests of Section 6.4 and 6.5. In each case, the sequence of failures and repairs is the same as in the corresponding test in Chapter VI.

The comparison is not entirely fair because the parameters of the operating rule are not optimized for each configuration tested. To optimize the parameters it would be necessary to perform simulation experiments with different decision rule parameter values. The experiments would then be repeated for each workcenter configuration.

B.1 The Decision Rule

1. For each part type, determine I^n_{max}, the maximum number of type n inprocess parts allowed inside the workcenter.
2. For each part type, a new piece is loaded into the system if (i) the production for the part trails demand, (ii) the number of type n inprocess parts is less then I^n_{max} and (iii) there is space in the workcenter for the new part.

The operating rule is easy to implement and incorporates feedback of buffer state information. Information on the machine state and workstation reliability is not required.

The best values of the parameters I^n_{max} and the demand rate to impose on the system would in practice be determined by experimentation. For the simulation runs described below, reasonable values of I^n_{max} are used. The demand rate is that analytically determined in Chapter VI using the equivalent production capacity set of Chapter V.

B. 2 Simulation Results

The operating rule was used to schedule the 2-part 3-stage system of Section 6.4. The demand requirements and processing times are those shown in Table 6.2, and the system parameters are given in Table 6.3. The maximum inprocess inventory, I^n_{max} for each part was 10.

Production statistics are shown in Table B.1. Total production at the end of the simulation run is comparable to that obtained using the EB flow control algorithm and given in Table 6.7. However, we note that the heuristic rule is biased towards the production of type 1 pieces. The average buffer state is 4.9 for type 1 pieces and -32.5 for type 2 compared to -8.2 and - 4.2 for type 1 and 2 pieces obtained by the EB flow controller. The reason for the bias is that the heuristic scheduling rule does not take into account the relative processing times for the two parts especially at the bottleneck stage A.

PART	REQUIREMENT	PRODUCTION	MEAN BUFFER STATE	MEAN INPROCESS INVENTORY	MEAN TRANSIT TIME
1	1104	1097	-4.9	4.1	1.79
2	552	531	-32.5	1.8	1.57

Table B. 1 Part Production for the Heuristic Operating Rule

The test was repeated for the 4-stage 4-part example of Section 6.5. The system parameters were those of Section 6.10, with internal storage buffers of size 10. For each part, I_{max} is 10. The demand requirement was that of Table 6.16 which is 95% of the equivalent production capacity.

Utilization of available time at the workstations, shown in Table B.2 is lower than that obtained with the EB controller and given in the bottom row of Table 6.17. The lower utilization is due to the fact that the heuristic rule does not hedge against machine failures. When production equals demand, the heuristic rule refrains from loading parts into the system. In the same situation, the EB controller loads parts at the maximum rate until the hedging point for the current machine state is reached. This has the effect of increasing workstation utilization and throughput.

The effect on production is shown in Table B.3. Compared to Table 6.19, we note that production is reduced by 146 pieces. The part production ratio is not close to the desired level because of a much lower production of type 4 pieces. This is due to the fact that type 4 pieces are more vulnerable to failures than other parts because they have the highest demand, require operations at all stages and they have relatively long pperation times at the bottleneck stages C and D.

The inprocess inventory and the mean transit times for the heuristic operating rule are comparable to those obtained by the EB controller. This is due to the effect of the ceilings I^n_{max} imposed on the inprocess inventory for each part.

STAGE	UTILIZATION OF AVAILABLE TIME
A	.68
B	.59
C	.90
D	.84

Table B.2 Workstation Utilization for the Heuristic Operating Rule

PART	REQUIREMENT	PARTS PRODUCED	% OF TOTAL PRODUCTION	AVERAGE INPROCESS INVENTORY	MEAN TRANSIT TIME (Minutes)
1	1852	1850	20	5.8	3.0
2	876	925	10	2.9	3.0
3	3521	3520	38	4.4	1.2
4	3521	2903	32	9.4	3.1
TOTAL	9770	9198	100	22.5	2.4

Table B.3 Part Production for the Heuristic Operating Rule

The heuristic operating rule can be improved by choosing values of I^n_{max} and hedging stocks for the parts in a way that accounts for the demand rate, relative part processing times and workstation reliability.

It would also be necessary to incorporate a priority scheme which clears backlogs of some parts before others. The EB flow controller uses the estimates of the optimal cost-to-go functions and the on-line linear program (4.20) to determine hedging points and part flowrates. It may also be possible to choose the best values of I^n_{max} and hedging points for a given system configuration using simulation experiments. However, we note that a single simulation run for the 4-part 4-stage example requires approximately 10 minutes of C.P.U. time on a Honeywell computer with the Multics operating system. A heuristic empirical approach to production control is therefore likely to be time consuming for workcenter personnel.

APPENDIX C

C.0 THE SIMULATION CODE

In this section, we give an overview of the essential features of the code that implements the simulation model of Chapter VI. We concentrate mainly on the implementation of the hierarchical controller and then briefly describe the support subroutines.

Section C.1 describes the data structure of the simulation code. This includes the representation of the parts, workstations and, internal and external storage buffers. The subroutines that implement the hierarchical controller are covered in Section C.2. A program listing is included.

Extensive use is made of features available in the PL/1 language under the MULTICS operating systems*. In particular, the use of structured variables and based variables with their associated pointers greatly simplifies the writing of the simulation code.

C.1 Workcenter Representation in the Simulation Code

The workcenter is represented by a set of variables which describe the state of the simulation model. In what follows, the variable names are in upper case and a brief description of the information stored in the variable is given in lower case.

Stage i in the system is represented by the structured array STAGE(i) which is declared as follows:

---

*MULTICS PL/1 Reference Manual, Honeywell Order No. AG94.

```
1    STAGE(i)
     2 FAILURE_RATE                stage i failure rate
     2 REPAIR_RATE                 stage i repair rate
     2 NO_OF_PROCESSORS            no. of Machines at stage i
     2 PROCESSOR(j)
          3 STATUS
            4 UP_TIME              total available time at stage i processor j
            4 STATE                stage i processor j status (e.g.,"FAILED)
            4 TIME                 Event clock for state change
          3 PART
            4 OP_TIME              total time actually processing parts
            4 LINK                 pointer to inprocess part at processor j
            4 TIME                 event clock for operation completion
```

The internal storage buffer for stage i is represented by the strutured array QUEUE(i):

```
1    QUEUE(i)
     2 CAPACITY                    buffer capacity
     2 NO_OF_PIECES                number of pieces in the buffer
     2 PART(q)
          3 LINK                   pointer to inprocess part in position q
                                   of the buffer
```

Part n data is in the structure array PART(n)

```
1    PART(n)
     2 DEMAND                      demand rate for type n parts (pieces/min)
```

2 INPROCESS_INVENTORY	number of pieces in the workcenter
2 BUFFER_LEVEL	number of pieces in the downstream buffer
2 LOADING_INTERVAL	current length of the loading interval
2 OPERATION(k)	
3 STAGE(i)	
4 PROCESSING_TIME	processing time for operation k at stage i for part n

Parts within the workcenter are represented by a based structure INPROCESS PART. The pointer to the variable is stored at the current location of the part.

1 INPROCESS_PART	
2 SER_NO	Unique serial number of each piece
2 TYPE	Part type
2 OPERATION(k)	
3 STATUS	Binary variable 1 if operation k is completed 0 otherwise

The load request event clocks are stored in the structure LOAD-REQUEST:

1 LOAD_REQUEST	
2 PART(n)	
3 TIME	event clock for load commands for type n pieces

Pending load request are stored in a linked token chain of based variables LPR, ordered according to latest time to load.

1 LPR	
2 LINK	pointer to the next token

2 PART_LINK pointer to inprocess part

C.2 Implementation of the Simulation Model

In this section, we give a brief description of the simulation code. First we describe subroutines that implement the hierarchical controller, then the action and support subroutines that impelement the various events that occur in the model, input and output data, and maintain statistics on the performance of the system. For the system of Chapter VI, the routing level is redundant. Hence only the flow and sequence control levels are discussed.

C.2.1 The Flow Control Level

Called by UPDATE_TIME and MACHINE_STATE_MONITOR. This subroutine activates the flow control level of the control hierarchy to calculate new throughput rates. A user set flag selects either the EB-flow controller or the CE flow controller. The default is the EB controller. Additional entry points FC_SET allows the user to select CE or EB controllers.

CE_CONTROLLER

Called by FLOW_CONTROLLER. Implements the CE flow controller. Given the current buffer state x and machine state i, it solves the mathematical programming problem (4.64)-(4.65) with

$$g(x) = g_{T_i}(x) = \max_n |x^n - y^n|$$

by a linear program using the IMSL** subroutine ZX3LP. The buffer stock y is chosen by the user and has a default value of y=0.

Additional entry point; CE_SET allows the user to set y.

EB_CONTROLLER

called by FLOW_CONTROLLER. Computes the throughput by the linear program (4.20) using the IMSL** subroutine ZX3LP. Quadratic estimates of the optimal cost-to-go must have been previously calculated.

CALCULATE_BOUNDS

Computes upper and lower bound of the optimal cost-to-go function by the successive approximations algorithm and stores them in the external variable COST-TO-GO.

QUADRATIC_FIT

Fits quadratic functions to bounds calculated by CALCULATE_BOUNDS. The coefficients are stored in external variables

Al(i,n,p), Bl(i,n,p) and Cl(i,n,p)

Sequence Control Subroutines

GENERATE_LOAD_REQUEST(n,t)

Called by EVENT_MONITOR. Generates a load request for a part of type n with a load window of width t and allocates an LPR variable with the latest time to load information.

**International Mathematical and Statistical Library (IMSL)

LOAD_PART

Called by EVENT_MONITOR. Scans the LPR load request chain and loads pieces into the stage with least number of waiting parts. An INPROCESS_ PART variable is allocated for each new part entering the system and the corresponding LPR variable is freed.

CANCEL_LOAD_REQUEST

Called by MACHINE_STATE_MONITOR and GENERATE_LOAD_REQUEST. Deletes from the LPR token chain any variable corresponding to an overdue load request. Additional entry point SET, allows the user to enable or disable the subroutine. The default is to have the subroutine enabled.

OPERATION_COMP(i,j)

Called by EVENT_MONITOR. Called when the operation at stage i processor j is completed. The actions taken are

1. Find a location for the next operation. If all operations are completed, the part is unloaded.
2. If all stages at which the part needs an operation are occupied, send the part to the central storage buffer.
3. For all blocked parts, check if a space has been created by the operation completion.
4. For any idle machine, check if there are pieces that need an operation at that machine.

The subroutine calls several other procedures in order to complete the actions.

Additional entry point OC(i,j) called by MACHINE_STATE_MONITOR performs actions 3 and 4.

NEXT_OP(PTR) RETURNS(i)

Called by OPERATION_COMP. For the inprocess part that is the object of the pointer PTR, the subroutine locates the stage i for the next operation. Among the stages at which the remaining operations can be performed, the one with the least number of waiting pieces is selected.

UNLOAD_PART(i,j)

Called by OPERATION_COMP. Unloads the part at stage i processor j and frees the INPROCESS_PART variable that is the object of the pointer STAGE(i).PROCESSOR(j).PART.LINK.

Support Subroutines

The simulation has several subroutines which perform various functions. They are briefly described below.

MOVE_PART 1 (i,j,n,m)

Called by OPERATION_COMP. Moves the part at stage i processor j to stage n processor m.

MOVE_PART 2 (i,j,n)

Called by OPERATION_COMP. Moves the part at stage i processor j to queue n.

MOVE_PART 3 (n,i,j)

Called by OPERATION_COMP. Moves the last part in queue n to stage i processor j.

MOVE_PART 4 (i,j)

Called by OPERATION_COMP. Moves the part at stage i processor j to the central storage area and creates a new variable STORAGE in the token chain.

MOVE_PART 5 (i,j)

Called by OPERATION_COMP and MOVE PART 4. Looks for a part in the central storage area that requires an operation at stage i. If one is found, the corresponding STORAGE variable is freed.

UPDATE_TIME(ql)

Called by EVENT_MONITOR. Decreaments all event clocks by the time ql elapsed since the last event.

INITIALIZE_SIMULATION

Called by SIMULATION. Initializes all variables in the program to the state corresponding to an empty system with all machines up.

DISPLAY

Called by SIMULATION. Gives a snap shot of the system by printing all system status information.

MACHINE_STATE_MONITOR(i,j)

Called by EVENT_MONITOR. Sets the correct status when a state change occurs at stage i processor j. Failure and repair events clocks are reset with a randomly generated number with the appropriate probability distribution.

FAILURE(i,j) RETURN(t)

Called by MACHINE_STATE_MONITOR. Generates and returns the time t to the next failure at stage i processor j when a repair is completed.

REPAIR(i,j) RETURN(t)

Called by MACHINE_STATE_MONITOR. Generates and returns the repair time when a failure event takes place at stage i processor j.

COSTS

Called by UPDATE_TIME. Updates production statistics data after each event. Additional entry point DISP called by DISPLAY prints the information.

EVENT_MONITOR(T)

Called by SIMULATION. Checks event clocks for the next event to occur in the system and then calls the relevant event subroutine. The first event after the final time T terminates the simulation.

SIMULATION

Main program, inputs system data and initiates the simulation run.

The following listing of the simulation code is provided for the readers information. The code is in a development stage and is therefore not completely debugged.

```
simulation: proc options (main);
dcl  initialize_simulation entry;
dcl  load_part entry;
dcl  generate_load_request entry (fixed, fixed);
dcl (p, r) float;
dcl  final_time fixed;
dcl (sysin, sysprint) file;
%include simulation_data_base;
dcl  event_monitor entry (fixed);
dcl  display entry;
dcl  top_level_controller entry;
dcl  machine_state_monitor entry (fixed, fixed);
dcl  buffer_monitor entry;
dcl  null builtin;
dcl  tt fixed;
dcl  time_step fixed;
dcl (i, j, k) fixed;
dcl  init bit (1);
          get list (print, init);
          put skip list (print, init);
          if init then do;
               clock = 0;
               get list (no_of_parts, no_of_stages, final_time, time_step);
               do i = 1 to no_of_stages;
                    get list (stage (i).no_of_processors);
                    get list (queue (i).capacity);
                    get list (p, r);
                    stage (i).failure_rate = p ;
                    stage (i).repair_rate = r ;
               end;
               do k = 1 to no_of_parts;
                    get list (part (k).demand);
                    do i = 1 to no_of_stages;
                         get list (part (k).operation (i).stage (i).processing_time);
                    end;
               end;
               call initialize_simulation;
               call top_level_controller;
               do k = 1 to no_of_parts;
                    tt = part (k).loading_interval;
                    call generate_load_request (k, tt);
               end;
               call load_part;

               call display;
          end;
          else get list (final_time, time_step);
          tt = clock + time_step;
          do while (tt <= final_time);
               call event_monitor (tt);
                call display;
               tt = tt+time_step;
```

```
    end;
end simulation;
```

```
cancel_load_request: proc;
%include simulation_data_base;
dcl  null builtin;
dcl  ptr pointer;
dcl  flag bit (1) static initial ("1"b);

          if flag then do while (lprptr ^= null);
               if lprptr -> lpr.time >= 0 then return;
               else;
               ptr = lprptr -> lpr.link;
               free lpr in (factory);
               lprptr = ptr;
          end;
          return;
set:      entry;
          get list (flag);
          if flag then put skip list ("cancel enabled");
          else put skip list ("cancel disabled");
          put skip;
          return;
     end;
```

```
ce_controller: proc;
dcl (q, q1, q2, r, p, i, j, k, l, m, m1, m2, n, ier, ia, iw (10000)) fixed bin (35);
dcl  tpint fixed static initial (60);
dcl (sysprint, sysin) file;
%include simulation_data_base;
dcl  cf float;
dcl (x (7), xstar (7), x1 (100), a1 (100, 100), b1 (100), c1 (100), lamda (15), et (15), rw (10000), y (100), s) float bin (27) exte
rnal;
dcl  flag bit (1) static initial ("1"b);
dcl  zx3lp entry options (variable);


          do k = 1 to no_of_parts;
               x (k) = part (k).buffer_level + part (k).inprocess_inventory - part (k).demand*clock/60;
          end;
          call set_up_lp;
          m1 = 2*p*no_of_parts + (p+1)*no_of_stages;
          m2 = 0;
          n = (p+1)*no_of_parts + p;
          ia = 100;
          put skip edit ("buffer state", (x (k) do k = 1 to no_of_parts))
               (a (12), 7 (e (12, 4), x (3)));
          call zx3lp (a1, ia, b1, c1, n, m1, m2, s, x1, y, rw, iw, ier);
          if print then put skip edit ((x1 (k) do k = 1 to (p+1)*no_of_parts+p)) (e (12, 4), x (3));
          put skip edit (ier, "control", (x1 (k) do k = p+1 to p+no_of_parts))
               (f (4), a (7), 7 (e (12, 4), x (3)));
          put skip;
          do k = 1 to no_of_parts;
               s = part (k).loading_interval;
               if x1 (k+p)>0 then part (k).loading_interval = 1/x1 (k+p);
               else part (k).loading_interval = 7200;
               load_request.part (k).time = max (0, part (k).loading_interval - s +load_request.part (k).time);
          end;
          return;
set_up_lp: proc;
               cf = 0;
               p = 1;
               do j = 1 to no_of_stages;
                    if alpha (j) < stage (j).no_of_processors then do;
                         p = p+1;
                         cf = cf + (stage (j).no_of_processors-alpha (j))* (1/stage (j).repair_rate);
                    end;
                    if alpha (j)> 0 then do;
                         p = p+1;
                         cf = cf+ alpha (j)* (1/stage (j).failure_rate);
                    end;
               end;
               do j = 1 to (p+1)*no_of_parts +p;
                    if j>p then c1 (j) = 0;
                    else c1 (j) = -1;
                    do k = 1 to (p+1)*no_of_stages + 2*p*no_of_parts;
                         a1 (j, k) = 0;
```

```
     end;
end;
et (1) = 1/cf;
lamda (1) = (1- cf * tpint);
k = 1;
do j = 1 to no_of_stages;
     if alpha (j)<stage (j).no_of_processors then do;
          k = k+1;
          et (k) = 1/ (cf + (1/stage (j).failure_rate)- (1/stage (j).repair_rate));
          lamda (k) = tpint* (stage (j).no_of_processors-alpha (j))/ (stage (j).repair_rate);
     end;
     if alpha (j)>0 then do;
          k = k+1;
          et (k) = 1/ (cf + (1/stage (j).repair_rate)- (1/stage (j).failure_rate));
          lamda (k) = tpint*alpha (j)/stage (j).failure_rate;
     end;
end;

q1 = p*no_of_parts;

do j = 1 to p;
     q = (j-1)*no_of_parts;
     do k = 1 to no_of_parts;
          a1 (j, q+k+q1), a1 (j, q+k) = -1/lamda (j);
          a1 (p+k, q+k) = -1;
          a1 (p+k, q+q1+k) = 1;
          a1 (q+k+p+no_of_parts, q+q1+k) = et (j);
          a1 (q+k+p+no_of_parts, q+k) = -et (j);
          b1 (q+k) = x (k)-xstar (k)- (1+et (j))*part (k).demand/tpint;
          b1 (q+k+q1) = -b1 (q+k);
     end;
end;

m = 1;
q2 = p+2*no_of_parts;
q1 = 2*p*no_of_parts+2*no_of_stages;
do j = 1 to no_of_stages;

     if alpha (j)<stage (j).no_of_processors then do;
          m = m+1;
          q = (m-2)*no_of_stages;
          r = (m-2)*no_of_parts;
          do k = 1 to no_of_stages;
               do l = 1 to no_of_parts;
                    a1 (r+q2+l, q+q1+k) = part (l).operation (k).stage (k).processing_time;
                    if k = j then b1 (q+q1+k) = alpha (k)+1;
                    else b1 (q+q1+k) = alpha (k);
               end;
          end;
     end;
     if alpha (j)>0 then do;
          m = m+1;
          q = (m-2)*no_of_stages;
          r = (m-2)*no_of_parts;
          do k = 1 to no_of_stages;
               do l = 1 to no_of_parts;
                    a1 (r+q2+l, q+q1+k) = part (l).operation (k).stage (k).processing_time;
                    if k = j then b1 (q+q1+k) = alpha (k)-1;
                    else b1 (q+q1+k) = alpha (k);
               end;
```

```
                    end;
               end;
          end;

          q2 = q2-no_of_parts*2;
          q1 = q1 -no_of_stages*2;
          do j = 1 to no_of_stages;
               do k = 1 to no_of_parts;
                    a1 (q2+k, q1+j) = part (k).operation (j).stage (j).processing_time;
                    a1 (q2+k+no_of_parts, q1+j+no_of_stages) = part (k).operation (j).stage (j).processing_time;
                    b1 (q1+ j), b1 (q1+j+no_of_stages) = alpha (j);
               end;
          end;

          if print then do;
               put skip edit ((c1 (j) do j = 1 to (p+1)*no_of_parts + p))
                    (f (7, 3));
               do j = 1 to 2* p*no_of_parts+ (p+1)*no_of_stages;
                    put skip edit ((a1 (k, j) do k = 1 to (p+1)*no_of_parts+p), b1 (j))
                         (f (7, 3));
               end;
          end;
          return;
     end set_up_ip;
ce_set:   entry;
put skip list(" input  no of parts, xstar and tpint");
get list (no_of_parts);
     get list ((xstar (j) do j = 1 to no_of_parts));
     get list (tpint);
     put skip list (" interval  =", tpint);
     put skip list ("xstar  ", (xstar (j) do j = 1 to no_of_parts));
     return;
end ce_controller;
```

```
costs: proc (q1);
dcl (q1, i, j, k) fixed;
%include simulation_data_base;
dcl  init bit (1) static initial ("1"b);
dcl (sysin, sysprint, dispsim) file;

dcl (bf (7), inp (7)) float static;
          if clock = 0 then init = "1"b;

          if init then do i = 1 to no_of_parts;
               bf (i) = 0;
               inp (i) = 0;
          end;
          init = "0"b;
          do i = 1 to no_of_parts;
               bf (i) = bf (i) + (part (i).buffer_level - part (i).demand*clock/60)*q1;

               inp (i) = inp (i) + part (i).inprocess_inventory*q1;
          end;
          return;
disp:     entry;
          if clock = 0 then return;
          put file (dispsim) skip edit ("part", "mean inprocess inventory", "mean buffer state")
               (a (4), x (3), a (23), x (3), a (17));
          do i = 1 to no_of_parts;
               put file (dispsim) skip edit (i, inp (i)/clock, bf (i)/clock) (f (4), x (7), e (12, 4), x (14), e (12, 4));
          end;
          return;
d_set:    entry;
          init = "1"b;
          return;
     end costs;
```

```
display: proc;
dcl (sysprint, dispsim) file;
%include simulation_data_base;
dcl  null builtin;
dcl (u, a) float;
dcl  ptr pointer;
dcl (i, j, k, n, m) fixed;
dcl  par fixed;
dcl  ser fixed;
dcl  costs$disp entry;
dcl 1 storage based (strptr),
    2 part_link pointer,
    2 link pointer;
dcl  strptr pointer external;
dcl  st_no_of_pieces fixed external;

          put file (dispsim) skip edit (("_" do i = 1 to 80)) (a (1));
          put file (dispsim) skip edit ("system state at time", clock) (x (30), a (20), f (7));
          put file (dispsim) skip (3);
          if lprptr ^= null then do;
               put file (dispsim) skip list ("pending load requests");
               ptr = lprptr;
               put file (dispsim) skip edit ("part", "latest time to load") (a (4), x (8), a (19));
               do while (ptr ^= null);
                    put file (dispsim) skip edit (ptr -> lpr.type, clock+ptr -> lpr.time) (f (2), x (10), f (7));
                    ptr = ptr -> lpr.link;
               end;
          end;

          if strptr ^= null then do;
               put file (dispsim) skip edit (" pieces in central storage area ")
                    (a (32));
               ptr = strptr;
               do while (ptr ^= null);
                    put file (dispsim) skip edit (ptr -> storage.part_link -> inprocess_part.type, "-", ptr -> storage.part_link ->
inprocess_part.ser_no)
                         (f (3), a (1), f (5));
                    ptr = ptr -> storage.link;
               end;
          end;

          put file (dispsim) skip edit ("machine state =", (alpha (i) do i = 1 to no_of_stages))
               (x (20), a (15), x (2), 5 (f (4)));
          put file (dispsim) skip edit ("----------------------------") (x (20), a (20));
          put file (dispsim) skip (2);

          do i = 1 to no_of_stages;
               put file (dispsim) skip edit ("stage", i) (x (10), a (5), f (3));
               put file (dispsim) skip edit ("--------") (x (10), a (8));
               put file (dispsim) skip edit ("processor", "status", "part", "time to finish", "utilization",
                    "availability") (a (9), x (2), a (6), x (5), a (4), x (7), a (14), x (1), a (11),
                    x (2), a (12));
```

```
            do j = 1 to stage (i).no_of_processors;
                 if stage (i).processor (j).status.up_time > 0 then
                      u = stage (i).processor (j).part.op_time/stage (i).processor (j).status.up_time;
                 else u = 0;
                 if clock > 0 then
                      a = stage (i).processor (j).status.up_time/clock;
                 else a = 1;
                 ptr = stage (i).processor (j).part.link;
                 if ptr = null then do ;
                      par = 0;
                      ser = 0;
                 end;
                 else do;
                      par = ptr -> inprocess_part.type;
                      ser = ptr -> inprocess_part.ser_no;
                 end;
                 put file (dispsim) skip edit (j, stage (i).processor (j).status.state, par, "-", ser,
                      stage (i).processor (j).part.time, u, a)
                      (f (7), x (4), a (9), x (2), f (1), a (1), f (5), x (4), f (7, 4), x (8), f (7, 4), x (7),
                      f (7, 4));
            end;
            put file (dispsim) skip;
            put file (dispsim) skip edit ("queue state") (x (10), a (11));
            put file (dispsim) skip edit ("-----------") (x (10), a (11));
            put file (dispsim) skip edit ("no of parts waiting", queue (i).no_of_pieces)
                 (a (19), f (4));
            put file (dispsim) skip edit ("pieces", (queue (i).part (k).link -> inprocess_part.type, "-", queue (i).part (k).link
 -> inprocess_part.ser_no
                 do k = 1 to queue (i).no_of_pieces)) (a (5),
                 20 (f (2), a (1), f (5), x (1)));
            put file (dispsim) skip (2);
        end;
        put file (dispsim) skip edit ("production statistics") (x (20), a (21));
        put file (dispsim) skip edit ("----------------------") (x (20), a (21));
        put file (dispsim) skip edit ("part", "inprocess_inventory", "buffer_level", "buffer_state")
             (a (4), x (3), a (19), x (10), a (12), x (12), a (12));
        do k = 1 to no_of_parts;
             a = part (k).buffer_level - part (k).demand*clock/60;
             put file (dispsim) skip edit (k, part (k).inprocess_inventory, part (k).buffer_level, a) (f (4), x (10), f (7), x (20
), f (7), x (15), e (12, 4));
        end;
        put file (dispsim) skip;
        call costs$disp;
        put file (dispsim) skip edit (("_" do i = 1 to 80)) (a (1));
        put file (dispsim) skip (2);
    end display;
```

```
eb_controller: proc;
dcl  sysprint file;
dcl  zx3lp entry options (variable);
dcl  a1 (10, 10) float bin (27);
dcl  rw (200) float bin (27);
dcl  u (7) float bin (27) static initial (0, 0, 0, 0, 0, 0, 0);
dcl (iw (200), ier, n, i, j, k, ia, m, m1, m2) fixed bin (35);
dcl (s, x (7), c1 (7), b1 (5)) float bin (27);
%include simulation_data_base;
dcl (a (7, 0:1000), b (7, 0:1000), c (7, 0:1000)) float external;
dcl (flag, flag1) bit (1);
dcl  ptr pointer;
dcl  null builtin;

          if print then put skip list ("eb controller calculating new production rates");
          do i = 1 to no_of_stages;
               do j = 1 to no_of_parts;
                    a1 (j, i) = part (j).operation (i).stage (i).processing_time;
               end;
          end;
          i = 0;
          m = 1;
          do j = 1 to no_of_stages;
               if j>1 then m = m* (stage (j-1).no_of_processors+1);
               i = i+ alpha (j)*m;
               b1 (j) = alpha (j);
          end;
          if print then put skip list ("machine state", (alpha (k) do k = 1 to no_of_stages), "i", i);
          do k = 1 to no_of_parts;
               x (k) = part (k).buffer_level + part (k).inprocess_inventory - part (k).demand*clock/60;
               c1 (k) = -2*a (k, i)* x (k) -b (k, i);

          end;
          m1 = no_of_stages;
          m2 = 0;
          ia = 10;
          n = no_of_parts;
          put skip edit ("buffer state", (x (j) do j = 1 to no_of_parts))
               (a (12), 7 (e (12, 4), x (3)));
          if print then put skip list ((c1 (j) do j = 1 to no_of_parts));
          flag = "1"b;
          do j = 1 to no_of_stages while (flag);
               s = 0;
               do k = 1 to no_of_parts;
                    rw (i) = a1 (k, j)* (-b (k, i)/ (2*a (k, i))-x (k)+part (k).demand)/60;
                    if rw (i) < 0 then flag = "0"b;
                    s = s + rw (i);
               end;
               if s>b1 (j) then flag = "0"b;
          end;
          if flag then do;
               do k = 1 to no_of_parts;
```

```
                u (k) = (-b (k, 1)/ (2*a (k, 1))-x (k)+part (k).demand)/60;
            end;
            ier = -111;
        end;
        else call zx3lp (a1, ia, b1, c1, n, m1, m2, s, u, x, rw, iw, ier);
        put skip edit (ier, "control", (u (i) do i = 1 to no_of_parts))
            (f (5), x (3), a (7), x (3), 7 (e (12, 4), x (3)));
        if print then put skip;
        do k = 1 to no_of_parts;
            s = part (k).loading_interval;
            if u (k)>0 then
                part (k).loading_interval = 1/u (k);
            else
            part (k).loading_interval = 3600;
            load_request.part (k).time = max (0, part (k).loading_interval - s + load_request.part (k).time);
        end;
end eb_controller;
```

```
event_monitor: proc (final_time);
dcl  sysprint file;
dcl  final_time fixed;
dcl  generate_load_request entry (fixed, fixed);
dcl  null builtin;
dcl  machine_state_monitor entry (fixed, fixed);
dcl  buffer_monitor entry;
dcl  top_level_controller entry;
%include simulation_data_base;
dcl  event char (16) varying;
dcl  move_part1 entry (fixed, fixed, fixed, fixed);
dcl  move_part2 entry (fixed, fixed, fixed);
dcl  move_part3 entry (fixed, fixed, fixed);
dcl  operation_comp entry (fixed, fixed);
dcl  load_part entry;
dcl  update_time entry (fixed);
dcl  display entry;
dcl (i, j, k, n, m) fixed;
dcl (q1, q2, q3, q4) fixed ;
dcl (p1, p2, p3, p4)float;
          do while (clock <= final_time);
               p1 = 999.0000;
               do m = 1 to no_of_stages;

                    do n = 1 to stage (m).no_of_processors;
                         if stage (m).processor (n).status.state = "operating" then
                              if p1 > stage (m).processor (n).part.time then do;
                                   p1 = stage (m).processor (n).part.time;
                                   i = m;
                                   j = n;
                                   event = "op comp";
                              end;
                         if stage (m).processor (n).status.time<p1 then do;
                              p1 = stage (m).processor (n).status.time;
                              i = m;
                              j = n;
                              event = "state chg";
                         end;
                    end;
               end;
               do k = 1 to no_of_parts;
                    if p1 > load_request.part (k).time then do;

                         p1 = load_request.part (k).time;
                         j = k;
                         event = "load rqt";
                    end;
               end;
               q1 = p1;
```

```
            call update_time (q1);
            if print then put skip list ("q1", q1, "event", event);
            if event = "op comp" then do;
                 call operation_comp (i, j);
                 call load_part;
            end;
            else if event = "load rqt" then do;
                 q2 = part (j).loading_interval;
                 call generate_load_request (j, q2);
                 call load_part;
                 call buffer_monitor;
            end;
            else if event = "state chg" then do;
                 call machine_state_monitor (i, j);
                 call top_level_controller;
            end;
       end;
       return;
  end event_monitor;
```

```
failure: proc(i,j) returns(fixed);
dcl (sysin,sysprint) file;
dcl s fixed bin(27) static initial (0);
dcl number float bin(27);
%include simulation_data_base;
dcl (i,j,k) fixed;
dcl random_$exponential entry(fixed bin(27),float bin(27));
if s = 0 then get list (s);
call random_$exponential (s , number);
return(number*stage(i).failure_rate);
end failure;
```

```
generate_load_request: proc (n, interval);
dcl  sysprint file;
dcl  null builtin;
%include simulation_data_base;
dcl  interval fixed ;
dcl (i, j, k, n, m)fixed;
dcl  ptr pointer;
dcl  ptr1 pointer, ptr2 pointer;
dcl  cancel_load_request entry;

          call cancel_load_request;
          if print then
               put skip edit ("generating load request, part", n, "  latest loading time",
               clock+interval) (a (29), f (3), a (21), f (7));
          ptr = lprptr;
          allocate lpr in (factory);
          lprptr -> lpr.time = interval;
          lprptr -> lpr.link = null;
          lprptr -> lpr.type = n;
          load_request.part (n).time = part (n).loading_interval;
          if ptr = null then return;
          ptr1 = ptr;
          ptr2 = ptr;
          do while ("1"b);
               if interval <ptr2 -> lpr.time then do;
                    lprptr -> lpr.link = ptr2;
                    if ptr2 = ptr then return;
                    ptr1 -> lpr.link = lprptr;
                    lprptr = ptr;
                    return;
               end;
               ptr1 = ptr2;
               ptr2 = ptr1 -> lpr.link;
               if ptr2 = null then do;
                    lprptr -> lpr.link = null;
                    ptr1 -> lpr.link = lprptr;
                    lprptr = ptr;
                    return;
               end;
          end;
          load_request.part (n).time = part (n).loading_interval;
     end generate_load_request;
```

```
initialize_simulation: proc;
dcl  null builtin;
dcl (i, j, k, n, m) fixed;
dcl  sysprint file;
dcl  failure entry (fixed, fixed) returns (fixed);
%include simulation_data_base;
dcl  strptr pointer external;
dcl  st_no_of_pieces fixed external;

          strptr = null;
          st_no_of_pieces = 0;
          lprptr = null;
          ipptr = null;
          do i = 1 to no_of_stages;
               alpha (i) = stage (i).no_of_processors;
               do j = 1 to stage (i).no_of_processors;
                    stage (i).processor (j).part.link = null;
                    stage (i).processor (j).part.link = null;
                    stage (i).processor (j).part.op_time = 0;
                    stage (i).processor (j).part.time = 0;
                    stage (i).processor (j).status.state = "idle";
                    stage (i).processor (j).status.time = failure (i, j);
                    stage (i).processor (j).status.up_time = 0;
               end;
               queue (i).no_of_pieces = 0;
               do j = 1 to 20;
                    queue (i).part (j).link = null;
               end;
          end;
          do k = 1 to no_of_parts;
               load_request.part (k).time = 0;
               part (k).inprocess_inventory = 0;
               part (k).buffer_level = 0;
          end;
          put skip list (" initial data");
          put skip edit ("stage", "no of processors", "failure rate", "repair rate")
               (a (5), x (15), a (16), x (4), a (12), x (8), a (11));
          do i = 1 to no_of_stages;
               put skip edit (i, stage (i).no_of_processors, stage (i).failure_rate, stage (i).repair_rate)
                    (f (5), x (20), f (5), x (9), e (12, 4), x (8), e (12, 4));
          end;
          put skip list ("part data");
          put skip edit ("part", "demand") (a (4), x (6), a (6));;
          do k = 1 to no_of_parts;
               put skip edit (k, part (k).demand) (2 (f (5, 2), x (6)));
          end;
          put skip list ("processing time matrix");
          put skip edit ("stage", "part") (a (5), x (20), a (4));
          do i = 1 to no_of_stages;
               put skip edit (i, (part (k).operation (i).stage (i).processing_time do k = 1 to
                    no_of_parts)) (f (4), x (16), 7 (x (3), f (7, 4)));
          end;
```

```
end initialize_simulation;
```

```
load_part: proc;
dcl  sysprint file;
dcl  null builtin;
%include simulation_data_base;
dcl (i, j, k, n, m, l, q) fixed;
dcl (ptr1, ptr2) pointer;
dcl (x, y, z, r, s, t) fixed (7, 4);
        ptr1 = lprptr;
        ptr2 = lprptr;
        do while (ptr2 ^= null);
             q = 20;
             do n = 1 to no_of_stages;
                  k = ptr2 -> lpr.type;
                  l = part (k).operation (n).stage (n).processing_time;
                  if l > 0 then
                       if q > queue (n).no_of_pieces then do;
                            q = queue (n).no_of_pieces;
                            m = n;
                       end;
                       else;
                  else;
             end;
             call lp;
        end;
        return;
lp:     proc;
             do j = 1 to stage (m).no_of_processors;
                  if stage (m).processor (j).status.state = "idle" then do;
                       call add_part;

                       stage (m).processor (j).status.state = "operating";
                       if print then
                            put skip edit ("dispatching part", k, "-", ipptr -> inprocess_part.ser_no, "into stage", m, " processo
r", j,"at time",clock)
                            (a (16), f (3), a (1), f (5), x (2), a (10), x (2), f (3), x (2), a (10), f (3),x(3),a(7),x(2),f(7));
                       stage (m).processor (j).part.time = part (k).operation (m).stage (m).processing_time;
                       stage (m).processor (j).part.link = ipptr;
                       return;
                  end;
             end;
             k = queue (m).no_of_pieces + 1;
             do while (k <= queue (m).capacity);
                  call add_part;
                  if print then
                       put skip edit ("loading part", i, "-", ipptr -> inprocess_part.ser_no, "into queue ", m)
                       (a (12), f (3), a (m), f (5), x (2), a (10), f (3));
                  queue (m).part (k).link = ipptr;
                  queue (m).no_of_pieces = k;
                  k = k+1;
                  return;
             end;
             ptr1 = ptr2;
```

```
                ptr2 = ptr2 -> lpr.link;
                return;
           end lp;
add_part: proc;
                allocate inprocess_part  ;
                i = ptr2 -> lpr.type;
                ipptr -> inprocess_part.ser_no = part (i).inprocess_inventory+part (i).buffer_level+1 ;
                ipptr -> inprocess_part.type = i;
                part (i).inprocess_inventory = part (i).inprocess_inventory+1;
                do q = 1 to no_of_stages;
                     ipptr -> inprocess_part.operation (q).status = "0"b;
                end;
                if ptr2 = lprptr then do;
                     ptr1 = ptr2 -> lpr.link;
                     ptr2 = ptr1;
                     free lprptr -> lpr  ;
                     lprptr = ptr2;
                end;
                else do;
                     ptr1 -> lpr.link = ptr2 -> lpr.link;
                     free ptr2 -> lpr  ;
                     ptr2 = ptr1 -> lpr.link;
                end;
                return;
           end add_part;
      end load_part;
```

```
machine_state_monitor: proc (i, j);
dcl  operation_comp$oc entry (fixed, fixed);
dcl  sysprint file;
dcl  null builtin;
%include simulation_data_base;
dcl (failure, repair) entry (fixed, fixed)returns (fixed);
dcl (i, j, k) fixed;
dcl cancel_load_request entry;
dcl  operation_comp entry (fixed, fixed);
dcl  load_part entry;

call cancel_load_request;
          if stage (i).processor (j).status.state ^= "failed" then do;
               put skip edit ("stage", i, "processor", j, "fails at time", clock)
                    (a (5), f (3), x (2), a (9), f (3), x (2), a (13), x (2), f (8));
               stage (i).processor (j).status.state = "failed";
               stage (i).processor (j).status.time = repair (i, j);
               alpha (i) = alpha (i)-1;
          end;
          else do;
               put skip edit ("stage", i, "processor", j, "up at time", clock)
                    (a (5), f (3), x (2), a (9), f (3), x (2), a (10), x (5), f (8));
               stage (i).processor (j).status.time = failure (i, j);
               alpha (i) = alpha (i)+1;
               if stage (i).processor (j).part.link ^= null then
                    stage (i).processor (j).status.state = "operating";

               else do;
                    stage (i).processor (j).status.state = "idle";
                    call operation_comp$oc (i, j);
               end;
          end;
          put skip edit ("machine_state =", (alpha (k) do k = 1 to no_of_stages))
               (a (15), x (2), 5 (f (5)));
     end machine_state_monitor;
```

```
 move_part1: proc (i, j, n, m);
dcl sysprint file;
dcl null builtin;
%include simulation_data_base;
dcl (i, j, n, m, k) fixed;
k = stage(i).processor(j).part.link-> inprocess_part.type;
if print then
put skip edit ("moving_part",k,"-",stage(i).processor(j).part.link->
inprocess_part.ser_no,"to stage",n,"processor",m,"at time",clock)
(a(11),f(3),a(1),f(5),x(2),a(8),f(3),x(2),a(9),f(3),x(3),a(7),x(2),f(7));
          stage (n).processor (m).part.link = stage (i).processor (j).part.link;
          stage (n).processor (m).status.state = "operating";
          stage (n).processor (m).part.time = part (k).operation (n).stage (n).processing_time;
          stage (i).processor (j).status.state = "idle";
          stage (i).processor (j).part.link = null;
          return;
     end move_part1;
```

```
move_part2: proc (i, j, n);
dcl  sysprint file;
dcl  null builtin;
%include simulation_data_base;
dcl (i, j, n, k) fixed;
          k = queue (n).no_of_pieces;
          queue (n).part (k+1).link = stage (i).processor (j).part.link;
          queue (n).no_of_pieces = k+1;
          if print then
               put skip edit ("moving part", stage (i).processor (j).part.link ->
               inprocess_part.type, "-", stage (i).processor (j).part.link -> inprocess_part.ser_no,
               "to queue ", n," at time ",clock)
               (a (11), f (3), a (1), f (5), x (2), a (9), f (3),a(9),f(7));
          stage (i).processor (j).part.link = null;
          stage (i).processor (j).status.state = "idle";
          return;
     end move_part2;
```

```
move_part3: proc (n, i, j);
dcl null builtin;
dcl  sysprint file;
%include simulation_data_base;
dcl (i, j, n, k) fixed;
          k = queue (n).no_of_pieces;
          stage (i).processor (j).part.link = queue (n).part (k).link;
          queue (n).part (k).link = null;
          queue (n).no_of_pieces = k-1;
          stage (i).processor (j).status.state = "operating";
          k = stage (i).processor (j).part.link -> inprocess_part.type;
          stage (i).processor (j).part.time = part (k).operation (i).stage (i).processing_time;
          if print then
               put skip edit ("loading stage", i, "processor", j, "with part", k, "-",
               stage (i).processor (j).part.link -> inprocess_part.ser_no," at time ",clock)
               (a (13), f (3), x (2), a (9), f (3), x (2), a (9), f (3), x (2), a (1), f (5),a(10),f(7));
          return;
     end move_part3;
```

```
move_part4: proc (i, j);
dcl (i, j, k, m, n, p, q) fixed;
dcl  move_part5 entry (fixed, fixed);
dcl 1 storage based (strptr),
    2 part_link pointer,
    2 link pointer;
dcl  strptr pointer external;
dcl  null builtin;
dcl  st_no_of_pieces fixed external;
%include simulation_data_base;
dcl  sysprint file;
dcl  ptr pointer;

if st_no_of_pieces > 100 then do;
stage(i).processor(j).status.state = "blocked";
return;
end;
          ptr = strptr;
          allocate storage;
          strptr -> storage.link = ptr;
          strptr -> storage.part_link = stage (i).processor (j).part.link;
          stage (i).processor (j).part.link = null;
          stage (i).processor (j).status.state = "idle";
          st_no_of_pieces = st_no_of_pieces+1;
          if print then
               put skip edit ("part ", strptr -> storage.part_link -> inprocess_part .type, "-", strptr -> storage.part_link -> inpr
ocess_part .ser_no, " moved to storage at time ", clock)
               (a (5), f (3), a (1), f (5), a (26), f (7));
          call move_part5 (i, j);
          return;
     end move_part4;
```

```
move_part5: proc (i, j);
dcl (i, j, k, m, n, p, s, q) fixed;
dcl 1 storage based (strptr),
    2 part_link pointer,
    2 link pointer;
dcl  strptr pointer external;
dcl (ptr, ptr1, ptr2) pointer;
dcl  next_op entry (pointer) returns (fixed);
%include simulation_data_base;
dcl  null builtin;
dcl  sysprint file;
dcl  st_no_of_pieces fixed external;

          if strptr = null then return;
          ptr1 = strptr;
          ptr2 = strptr;
          do while (ptr2 ^= null);
               ptr = ptr2 -> storage.part_link;
               if ^ptr -> inprocess_part.operation (i).status then do;
                    k = ptr -> inprocess_part.type;
                    s = ptr -> inprocess_part.ser_no;
                    if print then
                         put skip edit ("moving part", k, "-", s, " from storage to stage ", i, "processor ", j, " at time ", clock)
                         (a (12), f (3), a (1), f (5), a (23), f (3), a (10), f (3), a (9), f (7));
                    else;
                    stage (i).processor (j).part.link = ptr;
                    stage (i).processor (j).part.time = part (k).operation (i).stage (i).processing_time;
                    stage (i).processor (j).status.state = "operating";
                    if ptr2 = strptr then do;
                         ptr1, ptr2 = strptr -> storage.link;
                         free strptr -> storage;
                         strptr = ptr1;
                    end;
                    else do;
                         ptr1 -> storage.link = ptr2 -> storage.link ;
                         free ptr2 -> storage;
                         ptr2 = ptr1 -> storage.link;
                    end;
                    st_no_of_pieces = st_no_of_pieces - 1;
                    return;
               end;
               else do;
                    ptr1 = ptr2;
                    ptr2 = ptr1 -> storage.link;
               end;
          end;
          return;
     end move_part5;
```

```
next_op: proc (ptr) returns (fixed);
dcl (i, j, k, p, m, q, s, n) fixed;
dcl  ptr pointer;
%include simulation_data_base;
dcl flag bit(1);

          p = ptr -> inprocess_part.type;
          q = 20;
          flag = "1"b;
          do s = 1 to no_of_stages;
               m = part (p).operation (s).stage (s).processing_time;
               if m > 0 then
                    if ^ptr -> inprocess_part.operation (s).status then do;
                         flag = "0"b;
                         if q > queue (s).no_of_pieces then do;
                              q = queue (s).no_of_pieces;
                              n = s;
                              if q = 0 then do m = 1 to stage (n).no_of_processors;
                                   if stage (n).processor (m).status.state = "idle" then return (n);
                              end;
                              else;

                         end;
                         else;
                    end;
                    else;
               else;
          end;
          if flag then n = no_of_stages+1;
          else;
          return (n);
     end next_op;
```

```
operation_comp: proc (i, j);
dcl  sysprint file;
%include simulation_data_base;
dcl  null builtin;
dcl (i, j, k, n, m, p, q, s) fixed;
dcl  unload_part entry (fixed, fixed);
dcl  move_part1 entry (fixed, fixed, fixed, fixed);
dcl  move_part2 entry (fixed, fixed, fixed);
dcl  move_part3 entry (fixed, fixed, fixed);
dcl  move_part4 entry (fixed, fixed);
dcl  move_part5 entry (fixed, fixed);
dcl  next_op entry (pointer) returns (fixed);
dcl  flag bit (1);

          if print then
               put skip edit ("operation completed on part", stage (i).processor (j).part.link ->
               inprocess_part.type, "-", stage (i).processor (j).part.link -> inprocess_part.ser_no, "at stage", i, "processor", j)
               (a (27), f (3), a (1), f (5), x (2), a (8), f (3), x (2), a (9), f (3));
          stage (i).processor (j).part.link -> inprocess_part.operation (i).status = "1"b;
          n = next_op (stage (i).processor (j).part.link);
          if n > no_of_stages then call unload_part (i, j);
          else do;
               k = 1;
               do while (k <= stage (n).no_of_processors);
                    if stage (n).processor (k).status.state = "idle" then do;
                         call move_part1 (i, j, n, k);
                         k = stage (n).no_of_processors +1 ;
                    end;
                    else k = k+1;
               end;
               if stage (i).processor (j).status.state = "operating" then do;
                    if queue (n).no_of_pieces<queue (n).capacity
                    then call move_part2 (i, j, n);
                    else call move_part4 (i, j);
               end;
          end;
oc:       entry (i, j);
          do i = 1 to no_of_stages;
               do j = 1 to stage (i).no_of_processors;
if stage(i).processor(j).status.state = "blocked" then call move_part4(i,j);
else;
                    if stage (i).processor (j).status.state = "idle" then
                         if queue (i).no_of_pieces > 0 then
                              call move_part3 (i, i, j);
                         else call move_part5 (i, j);
                    else;
               end;
          end;
          return;
     end operation_comp;
```

```
repair: proc(i,j) returns(fixed);
dcl (sysprint,sysin) file;
dcl s fixed bin(27) static initial (0);
dcl number float bin(27);
%include simulation_data_b
dcl (i,j,k) fixed;
dcl random_$exponential entry(fixed bin(27),float bin(27));
if s = 0 then get list (s);
call random_$exponential (s , number);
return(number*stage(i).repair_rate);
end repair;
```

```
dcl  seed float external;
dcl  print bit (1) external;
dcl  factory area (4096) external;
dcl  lprptr pointer external;
dcl  ipptr pointer external;

dcl 1 stage (5) external,
    2 no_of_processors fixed,
    2 processor (5),
      3 status,
        4 up_time fixed,
        4 state char (16) varying,
        4 time fixed,
      3 part,
        4 op_time fixed,
        4 link pointer,
        4 time fixed,
    2 failure_rate float,
    2 repair_rate float;

dcl 1 queue (5) external,
    2 capacity fixed,
    2 no_of_pieces fixed,
    2 part (20),
      3 link pointer,
      3 operation;

dcl 1 part (7) external,
    2 demand float,
    2 inprocess_inventory fixed,
    2 buffer_level fixed,
    2 loading_interval float,
    2 operation (10),
      3 stage (5),
        4 processing_time float;

dcl 1 inprocess_part based (ipptr),
    2 ser_no fixed,
    2 type fixed,
    2 operation (10),
      3 status bit (1);

dcl 1 load_request external,
    2 part (7),
      3 time fixed ;

dcl 1 lpr based (lprptr),
    2 link pointer,
    2 type fixed,
    2 time fixed ;

dcl  no_of_parts fixed external;
```

```
dcl  no_of_stages fixed external;
dcl  clock fixed external;
dcl  alpha (5) fixed external;
```

```
top_level_controller: proc;
dcl (sysin, sysprint) file;
dcl  ce_controller entry;
dcl  eb_controller entry;
dcl  cont char (2) static initial ("eb");
%include simulation_data_base;
dcl i fixed;

          if cont = "eb" then call eb_controller;
          else if cont = "ce" then call ce_controller;
          else do i = 1 to no_of_parts;
               part (i).loading_interval = 60/part (i).demand;
          end;
          return;
tlc_set:  entry;
          put skip list ("which top level controller ?");
          put skip list (" ce => certainty equivalent  eb => estimate based ");
          get list (cont);
          if cont = "eb" then put skip list (" estimate based controller activated");
          else if cont = "ce" then put skip list ("certainty equivalent controller activated");
          else put skip list ("mt controll activated");
     end top_level_controller;
```

```
unload_part: proc (i, j);
dcl null builtin;
%include simulation_data_base;
dcl (i, j, k) fixed;
dcl  sysprint file;
dcl  ptr pointer;
          ptr = stage (i).processor (j).part.link;
          stage (i).processor (j).part.link = null;
          k = ptr -> inprocess_part.type;
          part (k).inprocess_inventory = part (k).inprocess_inventory-1;
          part (k).buffer_level = part (k).buffer_level+1;
          stage (i).processor (j).status.state = "idle";
          if print then
               put skip edit ("unloading_part", k, "-", ptr -> inprocess_part.ser_no)
               (a (14), f (3), a (1), f (5));
          free ptr -> inprocess_part ;
     end unload_part;
```

```
update_time: proc (q1);
dcl  top_level_controller entry;
dcl (sysin, sysprint) file;
dcl  tpint fixed static initial (60);
dcl  costs entry (fixed);
dcl  count fixed static initial (1);
dcl  null builtin;
%include simulation_data_base;
dcl  q1 fixed ;
dcl  ptr pointer;
dcl (i, j, k, n, m) fixed;
          if clock = 0 then count = 0;
          do i = 1 to no_of_stages;
               do j = 1 to stage (i).no_of_processors;
                    stage (i).processor (j).status.time = stage (i).processor (j).status.time-q1;
                    if stage (i).processor (j).status.state = "operating" then do;
                         stage (i).processor (j).part.time = stage (i).processor (j).part.time - q1;
                         stage (i).processor (j).part.op_time = stage (i).processor (j).part.op_time + q1;
                    end;
                    if stage (i).processor (j).status.state ^= "failed"
                    then stage (i).processor (j).status.up_time = stage (i).processor (j).status.up_time + q1;
               end;
          end;
          do k = 1 to no_of_parts;
               load_request.part (k).time = load_request.part (k).time - q1;
          end;
          ptr = lprptr;
          do while (ptr ^= null);
               ptr -> lpr.time = ptr -> lpr.time-q1;
               ptr = ptr -> lpr.link;
          end;
          clock = clock +q1;
          call costs (q1);
          if clock/tpint > count then do;
               call top_level_controller;
               count = count + 1;
          end;
          return;
ut_set:   entry;
          get list (tpint);
          put skip edit (" top level interval set to  ", tpint)
               (a (27), x (2), f (4));
          put skip;
          return;
     end update_time;
```

```
cb: proc options (main);
dcl  pri entry ((*) float, (*) float, (*, *) fixed dec (7, 5), (*) fixed);
dcl  fill entry ((*) fixed, (*, *) fixed decimal (7, 5));
dcl  flag bit (1);
dcl  update_cost entry ((*) fixed decimal (7, 4), fixed decimal (7, 4), fixed dec (7, 4), (*) fixed,
    (*)fixed, (*)fixed decimal (10, 5), (*, *)fixed decimal (7, 5), (*) float, (*)float, (*)float);
dcl  ck entry (fixed, (*) fixed, (*, *) fixed decimal (7, 5), (*) fixed) returns (bit (1));
dcl  eval_machine_state entry (fixed, (*) fixed, (*) fixed);
dcl  index entry (fixed, fixed, fixed) returns (fixed);
dcl  terminal_cost entry;
%include var1;
%include var2;
         discount_factor = 1;
                                                        /*  this section inputs problem parameters  */
         put skip list ("is this  a finite time, discounted cost or average expected value problem ?");
         put skip list ("enter 1, 2 or 3");
         get list (problem);
         if problem = 1 then do;
              put skip list (" input final time and time step length        ");
              get list (final_time, time_step);
         end;
         else if problem = 2 then do;
              put skip list (" input maximum no. of iterations, time_step and discount factor");
              get list (final_time, time_step, discount_factor);
         end;
         else do;
              put skip list (" input maximum no. of iterations and time step length");
              get list (final_time, time_step);
         end;
         put skip list ("no. of parts ?     ");
         get list (no_of_parts);
         put skip list ("no of processing stages?    ");
         get list (no_of_stages);
         machine_states = 1;

         put skip list ("do you want the upper or lower bound?");
         get list (bound);
         put file (results) data (no_of_stages, no_of_parts);
         put file (results)skip;


         do j = 1 to no_of_stages;
              put skip edit ("stage", j, "no. of processors, failure rate, repair rate") (a (7), f (3), x (3), a (45));
              put skip ;
              get list (no_of_processors (j), failure_rate (j), repair_rate (j));
              machine_states = machine_states* (no_of_processors (j) +1);
              do i = 1 to no_of_parts;
                   put skip edit ("processing time for part", i, " = ") (a (25), f (3), a (3));
                   get list (processing_time (i, j));
              end;
         end;
```

```
          dem = 0;
          do p = 1 to no_of_parts;
               average_cost (p) = 0;
               put skip edit ("demand rate for part", p, " = ") (a (20), f (6), a (3));
               get list (demand (p));
               dem = dem + demand (p);
               put file (results) data (demand (p));
          end;
          machine_states = machine_states -1;
          put file (results) skip edit ("size of machine state =", machine_states) (a (24), f (7));
          put file (results) skip;

          put file (results) skip list ("stage          no of processors   failure rate   repair rate");
          put file (results) skip edit ((i, no_of_processors (i), failure_rate (i), repair_rate (i)
               do i = 1 to no_of_stages)) (f (5), x (10), f (5), x (17), f (5, 3), x (8), f (5, 3), skip);
          put file (results) skip (2);
          put file (results) skip (2) list ("stage production rates ( taking reliability into account)");
          put skip;
          call pri (repair_rate, failure_rate, processing_time, no_of_processors);

          put skip ;
          do j = 1 to no_of_stages;
               failure_rate (j) = failure_rate (j)*time_step;
               repair_rate (j) = repair_rate (j)*time_step;
          end;
          do i = 1 to no_of_parts;
               demand (i) = demand (i)*time_step;
          end;
          dem = dem*time_step;
          put file (results) skip (2) list ("processing time matrix");
          put skip;
          do j = 1 to no_of_stages;
               put file (results) skip edit ((processing_time (i, j) do i = 1 to no_of_parts)) (f (7, 3));
          end;
          put skip;
          put skip list (" do you want to initialize the  cost to go  function to zero?");
          get list (initialize);
          put skip list ("do you want results printed after each iteration?      ");
          get list (print);


/* now the calculation of cost estimates starts  */
          do p = 1 to no_of_parts;
               put skip list ("part", p);
               put skip list ("enter xmin,xmax,x_step_size");
               get list (xmin, xmax, x_step_size);
               xmax = xmax/x_step_size;
               xmin = xmin/x_step_size;
               gs = xmax-xmin+1;
               kmid = floor ((xmin + xmax)/2);
               if initialize = "yes" then call terminal_cost;
               demp = demand (p)/x_step_size;
               put skip list ("the program is now calculating the cost estimates");
               if bound = "upper" then do;
                    do j = 1 to no_of_stages;
                         set (j) = 0;
                         do i = 1 to no_of_parts;
                              set (j) = set (j) + processing_time (i, j)*demand (i)/dem;
                         end;
                         set (j) = set (j)*x_step_size/time_step;
```

```
            end;
        end;
        else do;
            do j = 1 to no_of_stages;
                if processing_time (p, j)>0 then
                    set (j) = processing_time (p, j)*x_step_size/time_step;
                else set (j) = 99.99999;
            end;
        end;
        put skip data ((set (k) do k = 1 to no_of_stages));
        time = final_time-time_step;
        do while (time >= 0);
            if print then put skip edit ("time", time) (a (4), x (2), f (7, 3));
            call update_cost
                (demand, demp, dem, no_of_processors, alpha, set, processing_time, failure_rate,
                repair_rate, average_cost);
            if print then put skip (3);
            time = time - time_step;
        end;

        call fill (no_of_processors, processing_time);
        if ^print then do;
            put file (results) skip edit ("average cost = ", average_cost (p)) (a (15), e (11, 4));
            put file (results) skip list ("cost at initial time");
            put skip (3);
            put file (results) skip (3) edit ("part", p) (a (4), f (7));
            put skip;
            do i = 0 to machine_states;
                call eval_machine_state (i, alpha, no_of_processors);
                put file (results) skip edit ("machine_state", (alpha (j) do j = 1 to no_of_stages))(a (13), f (5));
                put file (results) skip edit ((cost_to_go (index (i, n, p)) do n = xmin to xmax))
                    (e (11, 4), x (1));
            end;
        end;
    end;
    put skip;


end
```

```
cost_function: proc (n1, px) returns (float);
dcl  px fixed;
%include var1;
dcl (x1, n1) float;
          x1 = n1*x_step_size;
          if x1>0 then q = x1;
          else q = -x1;
          return (q/100);
     end cost_function;
```

```
index: proc (i, n, k) returns (fixed);
%include var1;
          return (((k-1)* (machine_states+1)+i)*gs+n-xmin+1);
     end index;
```

```
update_cost: proc
               (demand, demp,dem, no_of_processors, alpha, set, processing_time, failure_rate, repair_rate,
               average_cost);
%include var1;
dcl  cost_function entry (float, fixed) returns (float);
dcl  eval_optimal_control entry ((*) float, fixed dec (7, 4), fixed, fixed dec (7, 4));
dcl  index entry (fixed, fixed, fixed) returns (fixed);
dcl  eval_machine_state entry (fixed, (*) fixed, (*) fixed);
dcl  ck entry (fixed, (*) fixed, (*, *) fixed dec (7, 5), (*) fixed) returns (bit (1));
dcl  total_cost (-30:30) float;
dcl  save_cost (0:1000, -20:20) float external;
dcl  flag bit (1);
%include var2;
          do j = 1 to no_of_stages;
               alpha (j) = 0;
          end;
          do i = 0 to machine_states;
               call eval_machine_state (i, alpha, no_of_processors);
               flag = ck (i, alpha, processing_time, no_of_processors);
               if flag then do;
                    do n = xmin to xmax;
                         q = 0;
                         nf = 1;
                         cf = 1;
                         do j = 1 to no_of_stages;
                              if alpha (j) > 0 then cf = cf-failure_rate (j);
                              if alpha (j)<no_of_processors (j) then cf = cf - repair_rate (j);
                              if j> 1 then nf = nf* (no_of_processors (j-1)+1);
                              if alpha (j)< no_of_processors (j) then
                                   q = q+ cost_to_go (index (i+nf, n, p))*discount_factor*repair_rate (j);
                              if alpha (j)>0 then
                                   q = q+cost_to_go (index (i-nf, n, p))*discount_factor*failure_rate (j);
                         end;
                         total_cost (n) = q + cost_to_go (index (i, n, p))*discount_factor*cf;
                    end;

                    call eval_constraint_set;
                    call eval_optimal_control (total_cost, demp, i, reachable_set);
                    if print  then do;
                         put skip edit ("machine state", (alpha (j) do j = 1 to no_of_stages))
                              (a (13), f (7));
                         put skip edit ((save_cost (i, n) do n = xmin to xmax))
                              (e (11, 4), x (1));
                    end;
               end;
          end;
          if problem = 1 then do;
               do i = 0 to machine_states;
                    do n = xmin to xmax;
                         cost_to_go (index (i, n, p)) = save_cost (i, n);
                    end;
               end;
```

```
          end;
          else call infinite_horizon;
          return;

infinite_horizon: proc;
               q1 = save_cost (machine_states, xmin)-cost_to_go (index (machine_states, xmin, p))-average_cost (p);
               q2 = q1;
               cf = save_cost (machine_states, kmid);
               do i = 0 to machine_states;
                    call eval_machine_state (i, alpha, no_of_processors);
                    flag = ck (i, alpha, processing_time, no_of_processors);
                    if flag then do;
                         do n = xmin to xmax;
                              q = save_cost (i, n)-cost_to_go (index (i, n, p))-average_cost (p);
                              if q1 < q then q1 = q;
                              else if q2 >q then q2 = q;
                              cost_to_go (index (i, n, p)) = save_cost (i, n)-cf;
                         end;
                    end;
               end;
               if mod (time, 10) = 0 then put skip list (cf, q1+average_cost (p), q2+average_cost (p));
               if problem = 3 then average_cost (p) = cf;
               return;
          end infinite_horizon;
eval_constraint_set: proc;
               q = 10**6;
               do j = 1 to no_of_stages;
                    if q> alpha (j)/set (j) then q = alpha (j)/set (j);
               end;
               if bound = "upper" then reachable_set = (q*demand (p)/dem);
               else reachable_set = q;
               return;
          end eval_constraint_set;


     end update_cost;
```

```
eval_machine_state: proc(i,alpha,no_of_processors);
%include var1;
%include var2;
do k = 1 to no_of_stages;
alpha(k) = 0;
end;
                        nf = i;
                        do j = 1 to no_of_stages;
                             alpha (j) = mod (nf, no_of_processors (j)+1);
                             nf = floor (nf/ (no_of_processors (j)+1));
                             if nf = 0 then return;
                        end;
                   end eval_machine_state;
```

```
dcl  cost_to_go (250000) float external;
dcl (p, xmin, xmax, xdmin, xdmax, gs, kmid) fixed external;
dcl (final_time, time_step, x_step_size, time)fixed decimal (7, 4)external;
dcl (i, j, k, m, n, nf, kl, ku, kstar) fixed ;
dcl (no_of_parts, no_of_stages)fixed external;
dcl (machine_states, problem) fixed external;
dcl (reachable_set, dem, u, u1) fixed decimal (7, 4);
dcl (bound,  initialize) char (8)external;
dcl print bit(1) external;
dcl (discount_factor) float external;
dcl (q, q1, q2, cf) float ;
dcl (sysprint, sysin)file;
```

```
dcl  demand (7) fixed decimal (7, 4);
dcl  demp fixed decimal (7, 4);
dcl  no_of_processors (5) fixed;
dcl  alpha (5) fixed;
dcl  set (5)fixed decimal (10, 5);
dcl  processing_time (7, 5)fixed decimal (7, 5);
dcl  failure_rate (5)float;
dcl  repair_rate (5)float;
dcl  average_cost (7) float ;
dcl  results file;
```

```
terminal_cost: proc;
%include var1;
          do i = (p-1)* (machine_states+1)*gs+1 to p* (machine_states+1)*gs ;
               cost_to_go (i) = 0;
          end;
     end terminal_cost;
```

```
ck:  proc (i, alpha, processing_time, no_of_processors) returns (bit (1));
%include var2;
%include var1;
dcl  flag bit (1);
          flag = "1"b;
          do k = 1 to no_of_stages;
               if alpha (k) = 0 then flag = "0"b;
               if processing_time (p, k) = 0 then
                    if alpha (k) ^= no_of_processors (k) then flag = "0"b;
          end;
          return (flag);
     end ck;
```

```
eval_optimal_control: proc (total_cost, demp, i,reachable_set);
dcl  total_cost (-30:30)float ;
dcl  save_cost (0:1000, -20:20) float external;
%include var1;
%include var2;
dcl  cost_function entry (float, fixed) returns (float);
          kstar = xmax;
          if reachable_set >0 then do;
               n = xmin;
               do while (n<xmax);
                    u = n;
                    q1 = time_step*cost_function (u/2, p);
                    q2 = cost_function ((u+1)/2, p)*time_step;
                    if q1+ total_cost (n)< q2+total_cost (n+1) then do;
                         kstar = n;
                         n = xmax;
                    end;
                    n = n+1;
               end;
          end;
          u1 = reachable_set - demp;
          do n = xmin to xmax;
               if n < kstar - u1 then u = n + u1;
               else if n>kstar+demp then u = n - demp;
               else u = kstar;
               k1 = floor (u);
               ku = k1+1;
               if k1<xmin then do;
                    cf = total_cost (xmin)-total_cost (xmin+1);
                    total_cost (k1) = total_cost (xmin) + cf* (xmin-k1);
                    if ku<xmin then total_cost (ku) = total_cost (k1)-cf;
               end;
               if ku>xmax then do;
                    cf = total_cost (xmax)-total_cost (xmax-1);
                    total_cost (ku) = total_cost (xmax)+cf* (ku-xmax);
                    if k1>xmax then total_cost (k1) = total_cost (ku) - cf;
               end;
               cf = u-k1;
               q1 = (total_cost (ku)-total_cost (k1));
               q2 = cost_function ((n+u)/2, p)*time_step;
               save_cost (i, n) = total_cost (k1)+q1*cf+q2;
          end;
          return;
     end eval_optimal_control;
```

```
pr1: proc (repair_rate, failure_rate, processing_time, no_of_processors);
%include var1;
%include var2;
          put file (results) skip edit ("part", (i do i = 1 to no_of_parts)) (column (no_of_parts*4), a (5), skip, x (3), (no_of_par
ts)f (8));
          put file (results) skip (2) list ("stage");
          do j = 1 to no_of_stages;
               put file (results) skip edit (j) (f (3));
               q1 = repair_rate (j)/failure_rate (j);
               q = no_of_processors (j)*q1/ (1+q1);
               do i = 1 to no_of_parts;
                    if processing_time (i, j)>0 then do;
                         u = q/processing_time (i, j);
                         put file (results) edit (u) (f (9, 4));
                    end;
                    else put file (results) list ("   --- ");
               end;
          end;
     end pr1;
```

```
save: proc options (main);
dcl sav file;
%include var1;
%include var2;
dcl ss(no_of_parts*(machine_states+1)*gs) float controlled;
allocate ss;
do i = 1 to no_of_parts*(machine_states+1)*gs;
ss(i) = cost_to_go (i);
end;
write file(sav) from(ss);
end save;
```

```
restore: proc options (main);
dcl sav file;
%include var1;
%include var2;
dcl ss(no_of_parts*(machine_states+1)*gs) float controlled;
get list (machine_states,no_of_parts,gs);
allocate ss;
read file(sav) into(ss);
do i = 1 to no_of_parts*(machine_states+1)*gs;
cost_to_go(i) = ss(i);
end;
end restore;
```

```
quadratic_fit: proc;
dcl (a (7, 0:1000), b (7, 0:1000), c (7, 0:1000)) float external;
dcl (t11, t12, t22, det) float;
%include var1;
dcl (x, y, z, xb, yb, zb, xy, xz, zy, xx, zz) float;
dcl  index entry (fixed, fixed, fixed) returns (fixed);
          put skip list ("input xmin,xmax,x step size");

          get list (xmin, xmax, x_step_size);
          xmin = xmin/x_step_size;
          xmax = xmax/x_step_size;
          put skip edit ("part", "state", "kstar", "a", "b", "c", "error")
               (a (4), x (4), a (5), x (4), a (5), x (10), 2 (a (1), x (15)), a (1), x (10), a (5));
          do p = 1 to no_of_parts;
               do i = 0 to machine_states;
                    kmid = xmin;
                    kstar = xmax;
                    b (p, i) = xmax*x_step_size;
                    do while (kmid<xmax);
                         if cost_to_go (index (i, kmid, p)) <= cost_to_go (index (i, kmid+1, p)) then do;
                              b (p, i) = kmid*x_step_size;
                              kstar = kmid;
                              kmid = xmax;
                         end;
                         kmid = kmid +1;
                    end;
                    if kstar>xmin then do;
                         xx, zz, xy, xz, zy = 0;
                         c (p, i) = cost_to_go (index (i, 0, p));
                         do n = xmin to kstar;
                              y = cost_to_go (index (i, n, p))- c (p, i);
                              z = n*x_step_size;
                              x = z*z;
                              xx = xx +x*x;
                              zz = zz +z*z;
                              xz = xz+ x*z;
                              xy = xy + x*y;
                              zy = zy + z*y;
                         end;
                         n = kstar - xmin +1;
                         x = xy;
                         det = xx*zz-xz*xz;
                         a (p, i) = (zz*xy-xz*zy)/det;
                         b (p, i) = (xx*zy -xz*xy)/det;
                         z = 0;
                         do n = xmin to kstar;
                              y = cost_to_go (index (i, n, p));
                              x = n*x_step_size;

                              y = (y - (x* (a (p, i)*x + b (p, i))+c (p, i)))**2;
                              z = z+y;
                         end;
```

```
                    z = z/ (kstar-xmin + 1);
                    put skip edit (p, i, kstar, a (p, i), b (p, i), c (p, i), z)
                        (f (4), x (3), f (4), x (7), f (4), 4 (x (3), e (12, 4)));
                end;
            end;
        end;
        put skip;
        return;
end quadratic_fit;
```

```
fill: proc (no_of_processors, processing_time);
dcl  index entry (fixed, fixed, fixed) returns (fixed);
dcl  eval_machine_state entry (fixed, (*)fixed, (*) fixed);
%include var1;
%include var2;
dcl  flag bit (1);

          do i = 0 to machine_states;
               call eval_machine_state (i, alpha, no_of_processors);
               flag = "0"b;
               do j = 1 to no_of_stages;
                    if processing_time (p, j) = 0 then do;
                         alpha (j) = no_of_processors (j);
                         flag = "1"b;
                    end;
                    else if alpha (j) = 0 then do;
                         alpha (j) = 1;
                         flag = "1"b;
                    end;
                    else;
               end;
               if flag then do;
                    k = 0;
                    m = 1;
                    do j = 1 to no_of_stages;
                         if j >1 then m = m* (no_of_processors (j-1)+1);
                         k = k+ m*alpha (j);
                    end;
                    do n = xmin to xmax;
                         cost_to_go (index (i, n, p)) = cost_to_go (index (k, n, p));
                    end;
               end;
          end;
          return;
     end fill;
```

```
save_ab: proc options (main);
dcl  abc file;
dcl  bbc file;
dcl (a (7, 0:1000), b (7, 0:1000)) float external;

          write file (abc) from (a);
          write file (bbc) from (b);
          return;
re_ab:    entry;
          read file (abc) into (a);
          read file (bbc) into (b);
          return;
     end save_ab;
```