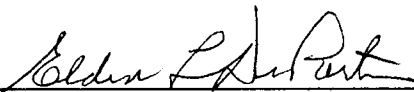


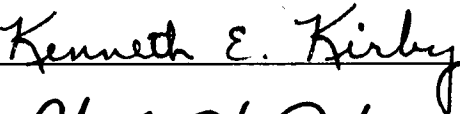
To the Graduate Council:

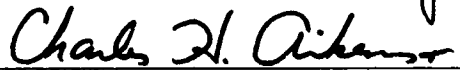
I am submitting herewith a dissertation written by Juan Miguel Sepulveda entitled "A Knowledge-based System for Real-Time Production Planning and Control in a Computer Integrated Manufacturing System." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Engineering Science.

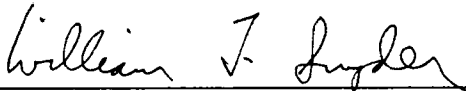

William G. Sullivan, Major Professor

We have read this dissertation
and recommend its acceptance:

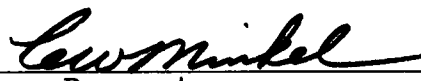








Accepted for the Council:


Vice Provost
and Dean of the Graduate School

A KNOWLEDGE-BASED SYSTEM FOR REAL-TIME PRODUCTION
PLANNING AND CONTROL IN A COMPUTER INTEGRATED
MANUFACTURING SYSTEM

A Dissertation
Presented for the
Doctor of Philosophy
Degree
The University of Tennessee, Knoxville

Juan Miguel Sepulveda
December 1988

DEDICATED

To my parents, Miguel and Amanda

ABSTRACT

During production, there is a need for real-time control and adaptive planning to minimize the effects of random events that affect the production capacity. Human shop floor managers must intervene and use their heuristic knowledge because the decisions are too unstructured and difficult to computerize. Knowledge-based systems are, in principle, an appropriate tool. Nevertheless, this is difficult because of the knowledge acquisition problem and the lack of suitable tools. This research investigates the use of model-based knowledge for real-time planning and control. A two-level hierarchical control system generates adaptive production trajectories using real-time information. The knowledge-based system monitors the production status, reacts to unplanned events, and performs replanning. The approach is demonstrated over a simulated cell for batch manufacturing with several workstations and material handling devices.

The results show that production goals are met with low variability and the system adapts well to temporary losses in capacity. Dynamic capacity allocation and dispatching achieve high equipment utilization. An adaptive routing algorithm, using real-time information about the traffic status, achieves low transportation delays. The shell designed for the experiments allows simple integration of heuristic and procedural knowledge for control of dynamic systems.

ACKNOWLEDGMENTS

I wish to express my gratitude to the faculty of the Industrial Engineering Department for their contributions over the course of my career. In particular, I am indebted to Professor William G. Sullivan, Chairman of the Committee, for introducing me into the field of this research, and for his subsequent guidance, criticisms, and support.

I owe special gratitude to Dr. Elden L. DePorter for his continuous support, valuable teachings, and critical review of this research. To Dr. John N. Snider, Head of the Industrial Engineering Department, for his participation and assistance over the years spent at the Department.

I wish to thank the members of the committee, Dr. Kenneth E. Kirby, Dr. Charles H. Aikens, and Dr. William T. Snyder, for their valuable comments and contributions.

Finally, I thank the authorities at the University of Santiago of Chile for allowing me to continue graduate studies and for their economic support.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION.....	1
1.1 Problem Statement.....	1
1.2 Objective and Purpose of the Research.....	5
1.3 Approach.....	7
1.4 Assumptions, Scope, and Limitations.....	10
1.5 Anticipated Results.....	12
1.6 Relevance of the Research.....	13
1.7 Overview.....	18
2. BACKGROUND ON KNOWLEDGE-BASED SYSTEMS.....	20
2.1 Artificial Intelligence and Expert Systems.....	20
2.1.1 Problem Solving.....	21
2.1.2 Knowledge Engineering.....	22
2.2 Heuristic Search.....	23
2.2.1 Blind Search.....	24
2.2.2 Informed Search.....	26
2.2.3 The A* Algorithm.....	28
2.3 Knowledge Representation.....	30
2.4 Methods to Represent Knowledge.....	31
2.4.1 Predicate Calculus.....	32
2.4.2 Production Rules.....	33
2.4.3 Probabilistic Reasoning.....	41
2.4.4 Approximate Reasoning.....	42
2.4.5 Semantic Networks, Frames and Scripts....	45
2.5 Selection of a KR Method.....	48
2.5.1 Tool Selection.....	51
3. LITERATURE SURVEY ON AI FOR ANALYSIS, PRODUCTION PLANNING, AND CONTROL IN MANUFACTURING SYSTEMS...	53
3.1 Industrial Applications of Knowledge-based Systems.....	53
3.2 Meta-modeling.....	57
3.2.1 Selection and Use of Models.....	58
3.2.2 Knowledge-based Simulation.....	60
3.3 Knowledge-Based Systems in Planning and Control.....	62
3.4 Progress of AI Research in Scheduling and Control.....	64
3.5 Development Environments for Manufacturing.....	69
3.6 Contributions of This Research.....	71

CHAPTER	PAGE
4. PRODUCTION PLANNING AND CONTROL IN COMPUTER INTEGRATED MANUFACTURING SYSTEMS.....	74
4.1 CIM, Production Planning, and Control.....	74
4.2 Schema for Planning and Control.....	77
4.2.1 Manufacturing Database.....	82
4.2.2 Hierarchical Integration.....	83
4.2.3 Computer Control Architectures	84
4.3 The Scheduling Problem.....	86
4.3.1 Level 1: Master Production Schedule (MPS)	87
4.3.2 Level 2: Material Requirements and Capacity Planning.....	87
4.3.3 Level 3: Shop Floor Scheduling.....	91
4.4 Combinatorial Methods for Short Term Scheduling.....	94
4.5 Methods to Reduce Complexity.....	96
4.5.1 Periodic Scheduling.....	96
4.5.2 Production Ratios.....	97
4.6 Dispatching Rules and Simulation.....	98
4.7 Control Theory.....	107
4.8 Summary.....	111
5. RESEARCH METHODOLOGY.....	115
5.1 Event-driven Inference.....	115
5.2 Architecture.....	117
5.3 Object Representation.....	121
5.4 Rule Representation.....	125
5.4.1 Variables and Directives.....	129
5.4.2 Top-level Algorithm for the CM Inference Engine.....	133
5.4.3 Computer Operating Systems and CM.....	135
5.5 Description of the Simulated Cell.....	136
5.5.1 Dispatching and Routing.....	137
5.5.2 Material Handling System.....	143
5.5.3 Random Factors.....	145
5.6 An Overview of the Experiments Performed with CM.....	145
6. EXPERIMENTS AND RESULTS.....	147
6.1 Deterministic Case.....	147
6.1.1 Trace of Sample Session.....	152
6.2 Examination Under Random Machine Failures.....	173
6.3 Analysis of Results.....	186
6.3.1 Analysis for the Mean Deviation.....	187
6.3.2 Effect on Utilization.....	196
6.3.3 Effect on the Length of the Run Time.....	196
6.3.4 Effect on the Station Buffers.....	196
6.3.5 Effect on the Time in the System.....	202

CHAPTER	PAGE
6. (Continued)	
6.4 Theoretical Justification and Approximate Formulas.....	202
6.4.1 Bound for the Station Utilization.....	207
6.4.2 Approximation for the Run Time.....	208
6.4.3 Approximation for the Time in the System.	209
7. CONCLUSIONS.....	211
7.1 Summary of the Methodology.....	211
7.2 Summary of Contributions.....	213
7.3 Recommendations for Further Research.....	216
LIST OF REFERENCES.....	218
APPENDIXES.....	239
A GLOSSARY.....	240
B A-STAR ALGORITHM.....	243
C KNOWLEDGE BASE LISTING.....	248
D SHORT USERS GUIDE.....	259
E FLOW-CHART FOR AGV SIMULATION.....	283
F PROGRAM LISTING.....	290
G DATA TABLES.....	291
H DERIVATION OF FORMULAS.....	320
VITA.....	322

LIST OF FIGURES

FIGURE	PAGE
1.1 Diagram of the Manufacturing Cell.....	6
2.1 Tree Generation in Heuristic Search.....	25
2.2 Backward- and Forward-chaining Inference in Rule-Based Systems.....	36
4.1 Main Functions of CIM Systems.....	76
4.2 Hierarchical Architecture of CIM systems.....	79
4.3 Measures of Performance in Scheduling.....	93
4.4 Example of Job Structure for Using Dispatching Rules.....	100
4.5 Description of Dispatching Rules.....	101
4.6 Indicators Used in Dispatching Rules.....	102
4.7 Due Date Calculation and Sorting Criteria.....	103
4.8 Characteristics of Dispatching Rules.....	104
5.1 Architecture of CM.....	118
5.2 Graphic View of Object Representation.....	122
5.3 Input Data and Production Rates in Example 6.3.	141
6.1 Object Definition for Example 6.1.....	149
6.2 Selected Traces in Example 6.1.....	154
6.3 Rules Fired in Example 6.1.....	158
6.4 Production Trajectory of Parts in Example 6.1..	162
6.5 Summary of Results in Example 6.1.....	165
6.6 Production Trajectory of Parts in Example 6.2..	169
6.7 Summary of Results in Example 6.2.....	172
6.8 Rules Used With Random Failures.....	174
6.9 Selected Trace in Example 6.3.....	177
6.10 Production Trajectory of Parts in Example 6.3.	181
6.11 Summary of Results in Example 6.3.....	184
6.12 Station Utilization Given MTR and MTBF Levels.	197
6.13 Length of the Run Time Given MTR and MTBF Levels.....	199
6.14 Buffer Size Given MTR and MTBF Levels.....	200
6.15 Time in the System Given MTR and MTBF Levels..	203
6.16 Change of Station State in Time.....	204
6.17 Plot of the (1/r) Factor.....	207
6.18 Plot of the r Factor.....	209

LIST OF TABLES

6.1	Parameter Levels for Experimental Design.....	188
6.2	Effect of Distribution Parameters on Mean Deviations.....	189
6.3	Confidence Intervals for Mean Deviations Given MRT Levels.....	190
6.4	Confidence Intervals for Mean Deviations Given MTBF Levels.....	191
6.5	Average Standard Deviation Given Levels of MRT and MTBF.....	193
6.6	Average Station Utilization and Run-Time Given Levels of MRT and MTBF.....	194
6.7	Average Buffer Size and Time in System Given Levels of MRT and MTBF.....	195

CHAPTER 1

INTRODUCTION

1.1 Problem Statement

In this research we address real-time production planning and control in a Computer Integrated Manufacturing (CIM) system.

The key function of real-time production control is to react promptly to random events that affect the production plan. Examples of these events are equipment failure, tool breakdown, shortage of parts, and changes in the demand. The objective of control is to develop actions which minimize the effects of such events. We call these actions "adaptive planning."

In real-time control the response time must be short. We want the lag between the detection of an event and the execution of proper actions to be minimal and attributable only to computation and data transmission.

Since 1970, manufacturing of discrete parts has undergone major innovations with the introduction of computer technology. Concepts and tools such as Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) are typical representatives of this change. CAM supports CIM with automated machines and material handling devices that process and handle parts under central or distributed computer control.

CIM has demonstrated its potential in the operational phase by flexible manufacturing systems (FMSs) and Local Area Networks (LANs). A LAN allows data transfer between shop floor devices and provides information systems with reliable, real-time data. The data is then available for the operators and managers to use it.

Yet, the basic problem of how to use the data for effective management remains as in conventional factories. Computer programs provide information, but it is the human operator who makes decisions. The operator decides when to load parts into the system, what part to load, how to assign machines and material handling devices, and how to manage emergencies such as machines failures and tool breakdowns.

Parallel to the effort of hard-automation is Artificial Intelligence (AI). Within AI, knowledge engineering is concerned with the representation and manipulation of knowledge so computers can emulate human problem solving. A knowledge-based system is a computer program built on knowledge engineering methods.

During production operations, a human operator may face many decisions during short periods. He uses a combination of heuristic knowledge and procedures to solve them. Knowledge-based systems can support real-time operations management when controllers (both humans and computerized) must concurrently handle a large amount of data from several sources and the response must be immediate.

High system variety is inherent to a CIM system. Hence, the effectiveness of control depends on the extent at which the controller matches the variety of the incoming information.

This research, therefore, focuses on real-time operations driven by a knowledge-based system. This approach is necessary for two basic reasons.

First, the variety problem above means that the manufacturing system can be in too many different states and each state requires a decision rule. A procedural solution would require full knowledge about those states before a complete algorithm is developed. In practice, such an implementation would be difficult to achieve because whenever an omitted state is discovered, or a rule needs to be modified, the algorithm would have to be re-programmed. On the other hand, a knowledge-based system has an inference engine which is independent of the rule base. Changes are made on the knowledge base but the engine remains unchanged.

Second, we can express heuristic rules (and even procedural knowledge by an appropriate representation) in declarative form. In general, declarative representation is easier to understand and modify than procedural representation. Hence, knowledge-based systems are easier to create and maintain, even by non-programmers.

In the production planning and control domain, however, the development of a knowledge-based system is complicated for several reasons:

1. Knowledge acquisition. For a knowledge-based system to be effective in production management, a thorough body of knowledge must be gathered and put into symbolic form for machine processing. Yet, this base of knowledge usually exists in a proprietary form, it is problem-specific, and facility-dependent.
2. Methodological differences. Knowledge-based systems need not rely only on heuristics. Mathematical and simulation models can, in some cases, provide information which is difficult to obtain from experts. However, models are procedural in nature. We must be able to transform or integrate this model-based knowledge into symbolic processing.
3. Lack of suitable models. The main difficulty of the model-based approach is that most of the existing research in this domain is about evaluative models for planning. Production scheduling models do not address real-time events that force re-planning. This is regarded as "too unstructured" for modeling. We need to find or formulate detailed models for real-time use and make a compromise between detail and computability.
4. Limitation of building tools. State-of-the-art tools have not yet achieved enough versatility to develop the

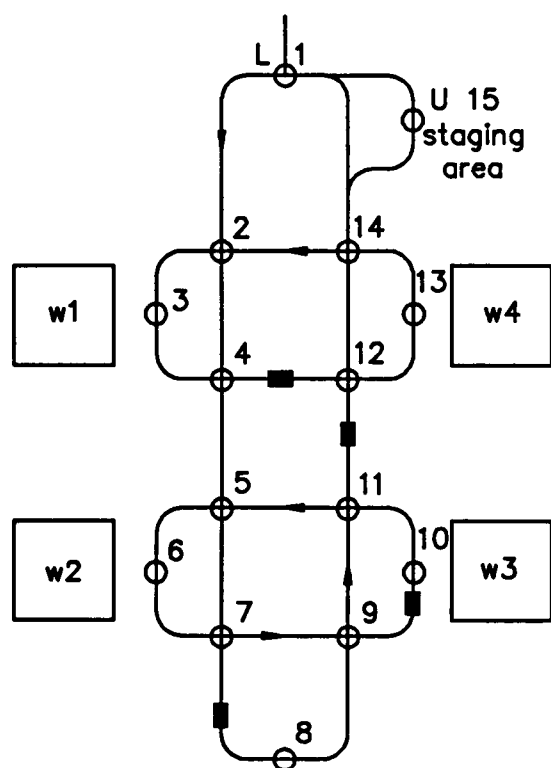
system pursued in this research. First, many existing applications, and by-product tools, have been developed on static domains. The correctness of the system is easily verifiable and the decisions do not have an impact over time. Second, these have been stand-alone systems which have not required access to procedural routines, special-purpose packages, or existing databases.

1.2 Objective and Purpose of the Research

The primary objective of the research is to build a knowledge-based system for production control in an automated manufacturing cell. Figure 1.1 shows the diagram of the cell. Henceforth, we refer to the system as the Cell Manager (CM).

The CM makes decisions in real-time. Given the requirements for processing a family of parts--as determined at higher levels of planning--the short-term planning and control tasks are the following:

1. To decide what parts to introduce in the cell, and at what time this occurs.
2. To assure the production requirements for the various parts are met even under foreseeable but random events such as machines breakdowns or other equipment failures.
3. To assign supporting resources such as machines and transporters (automated guided vehicles), and manage the flow of work pieces within the system.



Symbol

wn	workstation		vehicle
m	logical unit	L	loading dock
	m	U	unloading dock

Figure 1.1: Diagram of the Manufacturing Cell.

A secondary objective is to design a mechanism to study the CM's behavior in a dynamic system. We can regard the CM as a decision system conveying changes of state over a period time. As such, its decisions have an effect on the production system's performance.

To study the CM's behavior we use emulation. We build the CM as a knowledge-based system but the inputs (incoming data) and outputs (actions) are generated by, or performed onto a simulated device. In our case, this device is a simulation model of the facility. This approach is also useful to study the completeness of the control system, since we can detect in the design stage the conditions under which the system would fail because of incomplete knowledge.

The purpose of the research is to develop and test a methodology and a tool for implementing these systems. This includes determination of the information and communication needs, data and knowledge representation, and inference mechanisms for adaptive planning and control.

We design the software to implement CM because other expert system shells (development environments) lack of the capability to interact with mathematical models used in the research or cannot manage the dynamics of real-time events.

1.3 Approach

Any control system needs (a) control variables, (b) a planned trajectory for the variables, and (c) knowledge to maintain the variables close to the planned trajectory. The

latter involves algorithms and heuristics to react to both planned and random events that affect the trajectory.

For this purpose, we integrate two decisions levels. First, a control model determines the optimal trajectory of the production of parts over a period of time. Second, the CM controls the operations to meet the goals of the planning stage and generates feedback to correct the trajectory whenever it is necessary.

The optimal trajectory is to maintain the instantaneous production close to a reference level which minimizes work in process but at the same time makes provisions against temporary loss of capacity due to machine breakdowns. This is necessary in order to maintain a steady output rate to avoid starving other processes and/or cells.

Planning is adaptive. The trajectories must be re-planned whenever capacity losses arise or whenever the production is below or above its planned target level.

Thus, once CM is given the production targets, the parts are loaded into the cell according to a variable frequency derived from the control model. CM considers processing requirements, the status of the machines and transporters, and then passes the corresponding messages to the equipment controllers to coordinate the operations and satisfy the production targets.

The physical objects of the cell are the machines, the vehicles, the parts, the paths where the vehicles move, and

the points where these are located. We call these points logical units, as shown in Figure 1.1. These units can be machines, loading and unloading docks, or path interchange nodes.

Objects are represented as symbolic data structures containing several attributes. These attributes are for example the status of the machines, status and location of the vehicles, location and current operation of the parts, and the status of the paths onto which the vehicles move.

The status of the objects is dynamic. For example, machines and vehicles can be non-operative, idle, or assigned. Paths can be blocked or free. Parts are in different locations and stages of completion.

The attributes of the objects and the objects themselves (e.g., the parts in the cell) change over time according to the actions executed by the CM, or as a result of the objects' own change of state (e.g., when a machine fails or it is repaired).

At a given time, the cell and the parts in production are in a certain known state which generates a set of facts. These facts, in symbolic form, are considered by the CM's inference engine to decide what actions take place next.

The conditions and actions triggered by the facts are represented by predicates (if-then rules). The actions implemented by CM, in turn, alter the status of the objects,

and the system continues over time until the production is completed or until the execution horizon expires.

CM operates in two modes, to demonstrate how the system works and to measure its performance. In actual mode, the actions derived at any time are displayed as conclusions, as in conventional knowledge-based systems. These conclusions, in turn, are actions to be implemented on the cell by an installation dependent operating system. In the emulation mode, the actions modify the status of the objects and generate events and facts to be considered later in time in a closed loop form. However, the actions are performed onto the simulator and remain in the CM's working memory.

1.4 Assumptions, Scope, and Limitations

Since CM is intended for short-term scheduling and control at the shop floor, it is assumed that the production targets have been determined at a higher level in the planning hierarchy. These targets can be the net requirements specified by a Material Requirements Planning (MRP) approach, or by a mathematical model as in Hierarchical Production Planning (HPP). Hierarchical decomposition will be discussed in Chapter 4.

It is also assumed that the targets are realistic and within capacity. Capacity depends on how much time is available. Therefore, if the total requirements exceed the maximum possible output during the planning horizon, either

the lots have to split or a longer run is required. Again, these problems are assumed to be handled by higher levels of planning.

The study is oriented to batch manufacturing. However, the approach and tools to develop CM can apply to other types of manufacturing systems, such as continuous processes.

Although the cell is flexible, the parts are assumed to be similar enough so the machines can process different part types without incurring long setup times. This is possible by grouping lots according to shape or processing similarities. This grouping is done at a higher level by using, for instance, a Group Technology approach. This condition is necessary to guarantee that capacity can be rapidly redistributed among different parts within the family while the setup time consists only of adjusting, for instance, machining parameters in numerical control programs.

Additional assumptions, in particular those related to the emulation mode, include the time independence and statistical independence of machine failures and repair times. This does not affect generality since stochastic distributions, possibly interdependent, can be easily incorporated in the CM's knowledge base.

The research is limited to the emulation mode. In an actual application specific interfaces between the CM and lower-level control software need to be developed. The

knowledge base may then require extensions. However, the background and tools provided by the research serve as a basis for further applied investigation.

In an actual system, the CM would pass commands to the equipment controllers through a LAN. It is assumed that the LAN operates in real-time and transmissions delays are negligible. This is required in order for the CM to know the instantaneous status of the objects in the cell.

1.5 Anticipated Results

A shell (a development environment) for implementing the CM and its knowledge base is developed by using the VAX Common LISP dialect.

We simulate the cell and several production runs with machine failures to test the responsiveness of the CM. The actions of the CM are shown to be robust since it makes optimal allocation of the production capacity. This capacity is variable since it depends on the status of shop floor equipment. Despite of random capacity losses, the production goals are achieved by the due date (the end the planning horizon) with low variability. High equipment utilization is also achieved.

In most systems, a large fraction of the time is spent in transportation and handling. The CM coordinates machines and transporters to meet the production targets and minimizes the transportation delay by an adaptive vehicle routing

algorithm. This uses real-time information about the status of the paths. The system also controls the assignment of routes to avoid collisions.

Model- and heuristic-based knowledge are dynamic and change as the conditions of the facility and products are modified.

By using the CM's shell, variations in the decision rules, algorithms, and factory objects do not pose a heavy problem in re-designing the control software. Such changes only imply updates to the knowledge base. This is a major lesson and advantage of using a knowledge engineering approach.

As indicated, the design is evaluated over a hypothetical environment. But, it is done in such a manner that its actual implementation would involve the design of proper interfaces, to capture necessary data and to implement the actions prescribed. These are installation-dependent problems which the CM's shell makes provisions for, but they are subjected to further applied research.

1.6 Relevance of the Research

In the field of short-term scheduling a number of approaches have emerged from operations research and simulation. These essentially deal with the combinatorial nature of sequencing various activities that compete for the

same resources. This problem arises in job-shop and flow-shop type of manufacturing systems.

Although the models provide us with insight into this particular problem and also offer solutions under the assumption that the actual events will not alter the original schedule, the approaches are not oriented for implementation in real-time. Nor the available knowledge can be directly incorporated in an AI system.

Sequencing models can be formulated by integer linear programming and they are normally solved by using a heuristic which finds near-optimal solutions because a complete search is computationally expensive or impossible.

Heuristic search methods of AI can also be used to cut down the number of iterations to converge to a satisfactory solution. In this case AI is used to include in the evaluation function some specific knowledge about the scheduling problem. The heuristic function quantifies how promising a partial solution is, compared to the goal of scheduling all tasks at minimum cost (e.g., minimum completion time of all the operations, or makespan).

In the OR and AI cases, the models are static. They can be effective for planning but do not consider random events such as machine failures, delays in transportation, and processing times. These events occur during actual operation and therefore control and planning must be adaptive.

An alternative approach leading to more realistic modeling, in which random events can be easily included, is simulation. The simulation model is run under several dispatching rules which determine what activity follows next. These rules affect some performance measure such as the makespan, or the mean delay of the jobs with respect to their estimated due dates. The knowledge derived from the simulation results is later transferred to the actual system. That is, the scheduler would apply what was learned from the simulation. The rules can then be implemented in real-time. However, this has the inconvenience that a number of runs are required to tune the dispatching rules. The proper rules can be derived after a sufficient number of runs allows the deduction of statistically significant relationships (usually by experimental design followed by an ANOVA).

In any of three methods above, two key functions of control are missing: (a) the production targets at any given time, and (b) the decision rules to correct the behavior of the production when random events change the conditions of the system (in particular the capacity of the system).

To move into this problem area, we need to consider models of control theory. That is, endogenous variables such as the instantaneous production levels, are derived from the optimization of a constrained functional. The set of constraints may represent the instantaneous capacity of the

system which varies in time due to random factors. The constraints are systems of differential equations. The optimal trajectory given by the model provides the desired target.

In newer developments of control theory as applied to scheduling, researchers have found computable solutions for flexible production lines. That is, high-volume production lines for a family of parts where the machines can perform several different operations. Since production times are very small (order of seconds), the assumption of continuity holds and the delay between detection of a variation and the correction of the control variables is minimal (Akella, Choong, and Gershwin [5]). However, the approach has not yet been integrated on an AI-based controller, nor it addresses the case where longer production times (order of minutes) may exist.

Stecke [175], recognizes the need for AI in real-time control, but relatively little use of model-based knowledge has been done so far. One likely reason for this, is that conventional tools to build knowledge-based systems possess limitations to handle dynamic information and cannot be connected with the models. Another plausible explanation is that developers of expert systems are concentrated in coding "rules of thumb" obtained from human experts and little use of theoretical fundamentals is being made. Buchanan [32],

criticizes this common tendency in other domains and it must be avoided in this field.

As discussed in Chapter 3, there exist some case studies in which knowledge (e.g., properties and behavior) about the problem is derived from a model of the system. The tools developed in this research integrate some of these methodologies to perform real-time control with information that otherwise would be difficult (or unlikely) to obtain from experts.

The use of AI addressing real-time monitoring and control has been suggested but no full prototype including the inference engine and the knowledge base has emerged. Our research pursues such objective.

As contributions, the present research proposes and demonstrates a methodology to design an AI-based real-time control system using model-based knowledge in conjunction with heuristic knowledge for operational aspects where simple rules suffice. It shows the knowledge base utilized and the inference mechanism designed to handle the interaction with the control model. Also, a detailed emulation of the actions and events occurring in the cell is shown to demonstrate viability.

The results and research tools designed provide background and can be applied in similar cells, as shown in Figure 1.1.

1.7 Overview

In Chapter 2, we discuss some basic principles of artificial intelligence about heuristic search and knowledge representation.

In Chapter 3, we review important AI research as applied to various operational aspects of CIM systems, and also places the contribution of this research in perspective.

In Chapter 4, we describe the process of hierarchical planning and control in manufacturing systems. Although we are concerned with short-term scheduling and control, this bottom level is reached from higher levels of planning. In particular, it is covered the scheduling problem in its various planning horizons.

In Chapter 5, the methodology briefly presented in Section 1.3 is discussed in detail. A general view of the research tool to implement CM is presented. As indicated, the objects of the cell are characterized by dynamic properties or attributes. As the number of objects and their attributes grows, it is found that direct representation of facts in textual form is inefficient. To overcome this problem, the necessary information is kept in a database format and it is retrieved when the inference mechanism requires it. The rules support pattern-directed search. This means that the rules contain variables that are matched (instantiated) with the facts on the database. This technique was necessary to develop CM.

CM is developed in VAX Common LISP. PROLOG could also have been used. While PROLOG provides a built-in pattern-directed search mechanism and backtracking, LISP is more versatile to represent complex data structures and list manipulation. It also permits conventional programming and has interfaces to other languages in a manner less hardware-dependent than PROLOG.

The CM's inference engine is a problem-independent mechanism to deduce and implement actions on the cell. That is, the shop floor layout, events, facts, and decision rules become data. We need not re-program the engine.

At a given time, the status of the objects generate a set of facts which are filtered by the set of rules to decide what actions are implemented next at that time. This involves updating the attributes of the objects and performing actions such as loading a part, selecting a good path and moving a vehicle between two points, removing a part from a machine and taking it to another machine, redistributing capacity when equipment fails, etc.

In Chapter 6, experiments are conducted with the system by using the emulation mode and the events are traced through time. The knowledge base is shown and results from various scenarios are obtained. Theoretical foundations of the findings are examined.

Finally, in Chapter 7, comments are presented and topics for further research in the proposed methodology are discussed.

CHAPTER 2

BACKGROUND ON KNOWLEDGE-BASED SYSTEMS

The purpose of this chapter is to discuss the fundamentals of knowledge-based systems. We introduce the terminology and techniques that will allow us to develop the methodology of this research. The topics covered in this chapter will also let us examine the literature on AI for analysis, planning, and control of manufacturing systems.

2.1 Artificial Intelligence and Expert Systems

Artificial intelligence (AI) is the part of Computer Science concerned with designing intelligent computer systems. That is, systems which perform functions associated with human intelligence (Barr and Feigenbaum [10]). The functions include, for example, the human ability to (a) recognize images and scenes, (b) interpret messages from other humans, (c) modify the status of objects in the environment, (d) solve problems with unknown closed-form solution, and (e) acquire new knowledge, or modify it.

The approaches towards AI vary from psychological theories about human cognition to purely abstract constructions such as mathematical logic. The first resorts to models of cognitive psychology and linguistics. The second

relies on propositional and predicate calculus, and multi-valued logic.

Unlike physical sciences, however, the efforts in understanding the inner workings of the human mind have not yet produced concrete results and models. For this reason AI research is rather concerned with modeling "manifestations of human intelligence" (Kobsa [104]) from an external perspective. We have a perception about an intelligent behavior and create mechanisms to emulate it independently of its actual realization in mind.

Currently, AI research focuses on five general areas or domains which, with varying degrees of importance, emphasize one or more of the functions above. These areas are robotics, vision, natural language processing, problem solving, and learning.

We classify this research into the problem solving field. Therefore, our discussion will be centered on problem solving approaches. For a discussion on the other areas see for instance, Barr and Feigenbaum [10, Vol. 3], Rich [151], Shirai and Tsujii [168]. For specific languages and techniques to implement AI programs, see Bratko [30], and Winston and Horn [204].

2.1.1 Problem Solving

Problem solving deals with models of how humans find solutions to tasks requiring intellectual skills beyond the

simple application of an algorithm. This includes proof of theorems, solution to combinatorial problems, and use of available knowledge for deductive reasoning. This last problem is known as knowledge engineering.

2.1.2 Knowledge Engineering

Knowledge engineering (KE) is the activity concerned with the representation and manipulation of knowledge so a computer can solve unstructured problems. Computer programs in KE are known as knowledge-based systems. Their input are data structures storing knowledge available from different sources, such as books and human experts. Because of the latter and their expert-like behavior, these programs are also called expert systems (ESs). We can use either term indistinguishably.

The AI works in theorem proving and games in the mid fifties led to the idea of a general problem solving mechanism called production system. In this system, a problem is represented by a goal to be achieved, a state space of intermediate solutions, and a set of operators that transform the state space into new intermediate solutions towards the goal. The production system is independent of the specific domain, but an appropriate problem representation method allows the operators to exploit domain-specific knowledge to determine the path to the goal.

The mechanisms to control the production system gave origin to one of the two fundamentals of AI: heuristic search. The second is knowledge representation.

All areas of AI, in particular knowledge-based systems, rely on heuristic search and knowledge representation.

2.2 Heuristic Search

Heuristic search deals with combinatorial decision problems in which it is difficult, or simply impossible, to determine a path to a goal by a closed-form solution.

Examples of this kind can be found in:

1. Games such as chess, checkers, eight-puzzles, and maze searching.
2. Routing of a robot, vehicle, travelling salesperson.
3. Sequencing of operations in manufacturing systems.
4. Control of the inference in expert systems.

In heuristic search, a system (or object) change its state according to transition operators. The objective is to find a path (i.e., a sequence of operators) from the initial state to the goal. For instance, the states may be the status of the game, the location of the object, or the set of facts deduced at a certain point when interacting with a user. The goal may be to defeat the opponent, reach a destination point, have the part finished, or arrive to a conclusion.

Figure 2.1 depicts the search problem by a tree in which the state space are the nodes and the arcs represent the application of a production rule (operator). The new states are called the children of the parent node.

We can control the search by two general types of algorithms: blind and informed [151,168].

2.2.1 Blind Search

In blind search we reach the goal node by branching according to a deterministic pattern. Examples of this are the depth-first search and breadth-first algorithms.

In depth-first, we always expand a node, and check to see if it matches the goal, by the left-most child. In breadth-first, we generate the children level by level until the desired goal is found.

Blind search is simple and requires minimal information. However, it may be ineffective when we are interested in preferred solutions because it ignores measures of performance. For instance, in many combinatorial problems we can improve the search by exploiting specific knowledge about the state space and assign costs to the possible transitions. Alternative paths to the goal may arise, so we want the least cost path.

The two methods above would yield the first path found regardless the cost. There is no guarantee that either method will chose the best path. If the tree is large and

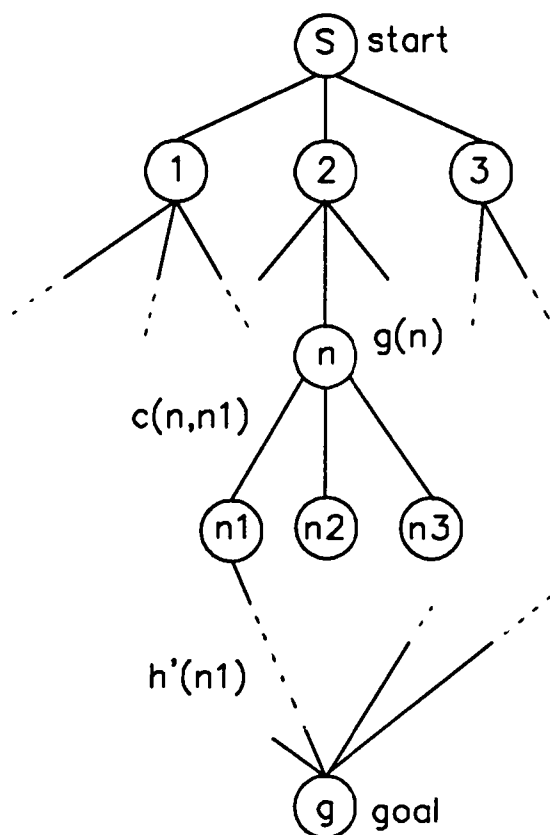


Figure 2.1: Tree Generation in Heuristic Search.

the goal is at a deep level, far to the right, these two methods would be inefficient due to the number iterations, or recursions, needed.

In other instances, the search does not requires a measure of performance. For example, we can use blind search methods to control the inference in a knowledge-based system, as discussed later in section 2.4.2.

2.2.2 Informed Search

In informed search the goal is reached by branching according to an evaluation function. This function keeps track of the cost from the initial state to the current node in expansion, plus an estimate of the cost to a goal.

There exist two main classes of algorithms. First, there are algorithms that use a minimax strategy, which is appropriate for some classes of games (e.g., chess). The Alpha-Beta cut-offs is an example of this class (see [10,151] for a description). Second, there exist algorithms to search trees and nets in order to minimize the cost from an initial node to a goal node. These optimal search methods are, in general, similar to those used in operations research.

The so-called ordered- or best-first search methods [10, 168] find "promising" paths by using "hill-climbing" (steepest-ascent, steepest-descent) techniques to move in the direction where the heuristic function improves most. Another algorithm, which we describe below, is the A* (pronounced A-

star) and uses a branch-and-bound (BAB) strategy to prune unsatisfactory paths.

The heuristic nature of AI methods arises from the difficulty in finding a well-defined mathematical function for the costs of the arcs, the heuristic function, or both. In some problems, it may be difficult to predetermine the cost of the arcs because the search space is too large; the space evolves as a result of exogenous factors; or the criteria to establish the cost measure is unclear.

A typical example difficult to solve due to the type of search space, is a chess game where the status of the board varies according to the strategies of each player. First, there may be several criteria to quantify the cost of a move (usually a weighted measure is used). Second, the heuristic function is only an estimate (or good guess) because of the number of subsequent moves made by the opponent.

An example of a changing state space found in this research is the routing of vehicles to move parts between logical units, as shown earlier in Figure 1.1. When routing a vehicle, the expansion of a node is the set of nodes a vehicle can visit next on the way to the destination. The arcs joining two nodes change their status due to external random factors. For instance, another vehicle might have entered an arc in the expansion, or the arc may be unavailable. These difficulties arise when the search is evolving.

Note that in the vehicle routing example, an integer

programming formulation for shortest path search would assume that all the paths are free as a vehicle moves from the start to the end point. Since the status of the arcs change while a vehicle is moving (other vehicles use the paths and queuing could occur), the travel time is not deterministic. We may achieve a shorter travel time by using a heuristic cost function which estimates, in real-time, how much would it cost to move according to the present status of the arcs on the traffic network.

A detailed statement of A* can be found in [10,151,168]. Here we summarize the mechanics and properties. Appendix B includes a LISP function to implement it.

2.2.3 The A* Algorithm

Let us define the following notation:

- n = current node in the search (Figure 2.1),
- $c(i,j)$ = cost of the edge connecting node i and node j ,
- $g(n)$ = cost of getting from the start node to node n ,
- $h'(n)$ = estimated cost of getting from n to the goal, where $h(n)$ is the actual value,
- $f'(n)$ = $g(n) + h'(n)$, where $f(n)$ is the actual value (the actual value is known after the optimal path has been found),

$O(n)$ = operator or production rule generating the set of successor states than can be derived from node state n .

Given a current node n , $O(n)$ generates the set of successors S . For each s in S , $f'(s)$ is estimated by:

$$f'(s) = g(n) + c(n,s) + h'(s).$$

The search proceeds by bringing all the new candidates into a sorted list in increasing order of f' and by setting a pointer from s to n . A node s already generated needs a pointer update. If the new $f'(s)$ is better than its old value then it will point to n , else the old s is put back in the candidate list and the search proceeds. This method to assign pointers to the parent node prevents the search from cycling when the expansion generates a node already examined.

Two important results due to Nilsson [168] establish when the algorithm converges and when the solution is optimal.

Nilsson's theorems.

Theorem 1:

If there exists a path from the starting node to the goal node, then the A* algorithm (2.1) converges in a finite number of steps.

Theorem 2:

If Theorem 1 holds, $0 \leq h'(n) \leq h(n)$, and
 $c(i,j) \geq 0$ for all i,j , then the A* algorithm (2.2)
converges and the solution is optimal.

Problems such the job-machine assignment, travelling salesperson, flow-shop sequencing, and job-shop sequencing, can be solved by the A* algorithm and also by integer programming. By theorems (2.1) and (2.2), A* yields the same result as the OR method solved by branch-and-bound.

2.3 Knowledge Representation

The areas of AI mentioned in section 2.1 have their own purpose and methods. Their common ground is that all of them are driven by knowledge about the specific domain. We refer the domain as the "world."

The notion of knowledge refers to:

1. Structure, form, properties, functions, and possible states of the world's objects.
2. Possible relationships between objects and the events in which they can participate.
3. Possible effects of events and preconditions that trigger these events and states.
4. Possible goals and plans.
5. Extent at which the knowledge is certain or credible.

Knowledge representation (KR) deals with (a) the design of syntactic mechanisms to represent facts and truths about the world, and (b) design of methods to draw inferences from them.

A KR mechanism is divided into two basic parts: the knowledge base and the inference engine. The general principle is that the knowledge base contains domain-specific knowledge in declarative form. The inference engine is a domain-independent search mechanism to infer facts from the knowledge base.

2.4 Methods to Represent Knowledge

There exists several approaches to KR. The selection of a particular method depends on the characteristics of the knowledge encoded. In this section we review these methods to discuss the approach followed in the research and to examine related works.

One KR method, rule-based representation, is automatically associated with ESSs because most of the early applications, and subsequent commercial products, have used it. Efforts based on theories of cognition, in particular the idea of frames covered in section 2.4.5, have not overcome its wide use. It is interesting to note that rules are theoretically founded on predicate calculus of first order, and not on cognitive theories.

KR methods fall into one or more of the following categories: predicate calculus, production rules, probabilistic reasoning, approximate reasoning, and frames.

2.4.1 Predicate Calculus

This classical declarative KR method is important as a theoretical foundation for rule-based representation. It is also the basis of an important development in logical programming, PROLOG (see Bratko [204] for AI programming).

The knowledge base consists of a set of propositions about objects. Axioms are translated into logical formulas that constitute the basis of a theory. The axioms are truths by which new theorems can be constructed. In predicate calculus, the inference is carried out by formal methods used in theorem proving. One of these methods is Robinson's Principle of Resolution to implement what is commonly known as "proof by refutation" [151,168].

To illustrate the use of predicate calculus, let us suppose we need to express knowledge to recognize when an object X is a rectangle (in the world of flat geometrical figures). We could write the axioms as the following formulas:

$\forall X, \text{has-four-sides}(X) \rightarrow \text{quadrilateral}(X).$

$\forall X, \text{quadrilateral}(X) \wedge \text{has-a-pair-of-opposite-sides-parallel}(X) \rightarrow \text{trapezoid}(X).$

```

∀X, trapezoid(X) ∧ base-angles-equal(X) -->
    isosceles-trapezoid(X).

∀X, trapezoid(X) ∧ has-a-second-pair-of-sides-
    parallel(X) --> parallelogram(X).

∀X, [isosceles-trapezoid(X) ∧ has-a-right-angle(X)] ∨
    [parallelogram(X) ∧ has-a-right-angle(X)] -->
    rectangle(X).

```

The atomic formulas (e.g., `has-four-sides(X)`) are the predicates. The symbol ' $\forall$ ' is the universal quantifier, read as 'for all.' The symbols ' $\wedge$ ' and ' $\vee$ ' stand for the conjunctive (and) and disjunctive (or) operators, respectively. The symbol ' $\rightarrow$ ' reads 'implies' and the inference from left to right is called 'modus ponens.'

2.4.2 Production Rules

In this method we represent knowledge by rules of the type:

```

if
    fact-1 and fact-2 and....or fact-n                (2.3)
then
    conclusion-1...and conclusion-m,

```

which reads "if fact-1 is present and fact-2 is present... or fact-n is present, then conclusion-1 is a fact...and conclusion-m is a fact."

The components of the 'if' and 'then' parts are syntactically equivalent to predicates. Predicates on the left-hand-side are called antecedents and the predicates on the right-hand-side are called consequents.

By using variables in the antecedents and consequents, rules are analogous to formulas of predicate calculus.

Note that rules should not be interpreted as branching logic as in programming languages but as statements about facts.

Two common control methods used in directing the inference in rule-based systems are (a) backward-chaining and (b) forward-chaining.

Backward-chaining. In backward-chaining we start by choosing the consequent to be proven and check whether the hypotheses hold. Each hypothesis is a goal to be reached. A goal can be the consequent of another rule. Then the strategy to prove a goal is recursive. To visualize this, we can depict the search space by an AND/OR tree. The nodes are the predicates of the rules and the arcs connect the antecedents with the consequents in a form consistent with the rules.

For example, suppose we have the following simple set of rules with the P's denoting predicates:

if P1 then P2

if P2 and P3 or P4 and P5 then P6.

Suppose we want to prove P6. By using depth-first control, we first attempt to prove P2. The rule for P2 fires if P1 is a fact (a rule is said "to fire" if all its antecedents are true). Once P2 is true, P3 must be proven. If P3 is true the second rule fires and P6 becomes a fact. In the event that P2 or P3 cannot be proven then P4 and P5 are attempted. If P4 and P5 are true then the second rule fires, else the top goal P6 cannot be deduced (inconclusive). This is illustrated in Figure 2.2 (a).

A generalization of the example above is as follows:

Step 1. While there is a top goal to prove, do:

- (a) If the current goal is a fact then goal is true, exit.
- (b) Else, if there exists a rule whose consequent contains the goal then invoke this procedure recursively to prove each rule antecedent. If all antecedents are true then assert that goal is true, exit.
- (c) Else, if the goal can be asked and the user asserts that the goal is true, exit.
- (d) Else, select a new goal and repeat from Step 1 above.

Step2. If no goal is left, fail.

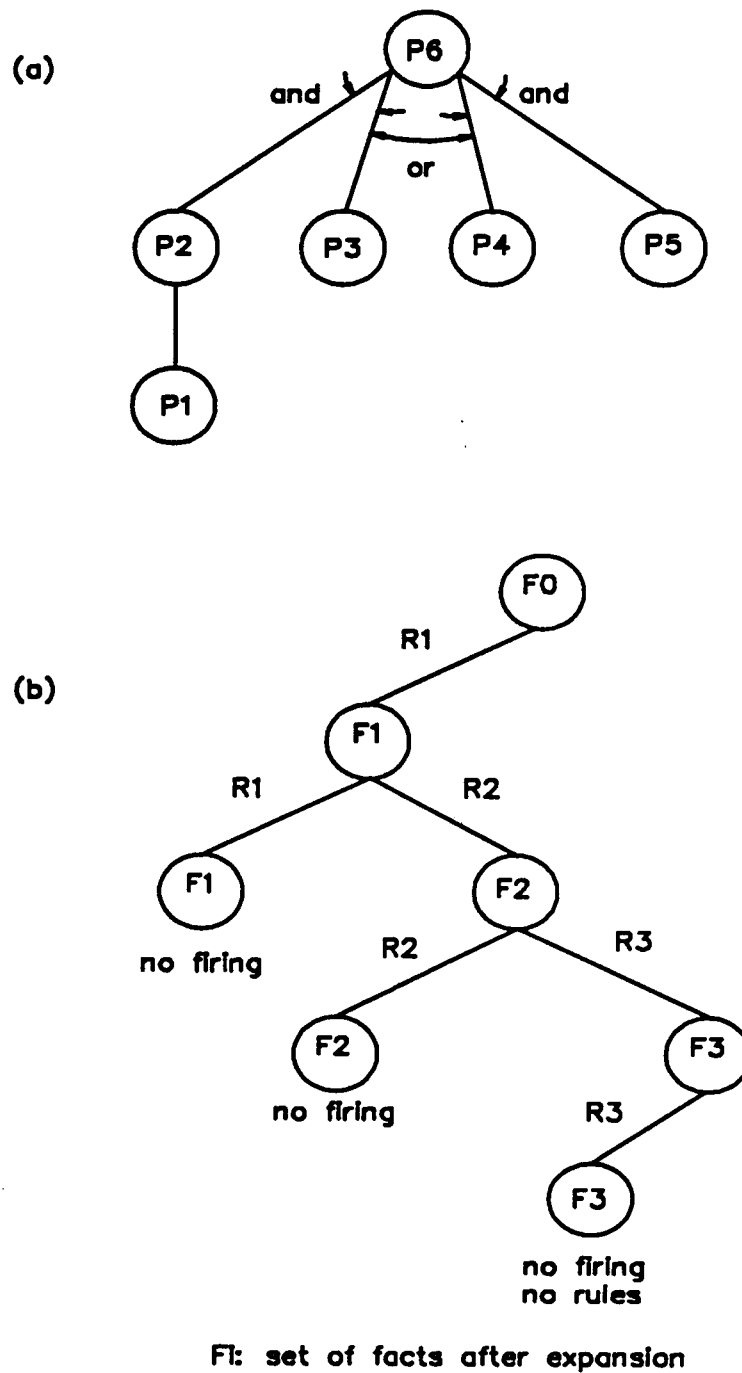


Figure 2.2: Backward- and Forward-Chaining Inference
in Rule-Based Systems.

Note that in the example of Figure 2.2 (a), the inference would not allow changing a fact stated as false by the user once a goal has been dropped out. As such, the reasoning is said to be monotonic, as opposed to non-monotonic reasoning where facts can be retracted to redo the proof.

Forward-chaining. In forward-chaining, we start from an existing set of nodes and traverse the rules expanding the nodes into new facts. Whenever a rule fires, the set of facts is expanded by asserting the new consequents. The search then iterates according to a control scheme. For instance, if given the current set of facts a rule fires, new assertions are added to the database (working memory) and the same rule tried again. If the rule fails, then a second is tried, and so on. Whenever a rule asserts a new fact, the rules are reset because the new evidence could serve to trigger rules previously discarded. Eventually, the search ends when nothing new can be deduced. This is illustrated in Figure 2.2 (b) for a simple case with three rules.

Meta-knowledge and procedural considerations. Although the control methods above provide the basic elements to guide the search, a higher level of control may be necessary. For instance:

1. To select the order in which the program ask questions.

2. To a priori prune unwanted branches in the search space.
3. To resolve conflicts arising from multiple firings.

In the first case, suppose an ES in a fire safety domain must guide a person to handle the emergency by asking whether people is in the facility. A rule may be as follows:

```
if  (there is a fire) and
    (no people is in the facility)
then (release the extinguisher)
```

In this rule there is an implicit procedural ordering of the antecedents. It would make little sense to ask about the second fact before the first.

In the second case, suppose an ES helps design the process plan for a cylindrical part. A very large knowledge base may exists for all geometric shapes and the base has been partitioned into several independent sub-bases. Then, we need a directive to indicate the ES to use only the corresponding partition. This is achieved by using meta-rules (or rule on rules). For instance:

```
if  (part is cylindrical)
then (load rules for cylindrical parts only).
```

In the third case, we may have a large knowledge base built incrementally. Two different rules may have the same antecedents but different consequents. One rule may be more important than the other, depending on the "line of reasoning" of the expert, and we want the rules fired in a given sequence. The control mechanism must then use a conflict resolution method which is specified by a procedural insertion. For example, we can associate priority indicators to the rules, as the "salience" of Clayton [41].

The value of the priorities can be simply motivated by the desired behavior of the inference but it may entail a more elaborated approach. For instance, when the problem involves conflicting objectives (e.g., a design problem where cost and safety are conflicting) we would use those rules that provide the preferred solution according to the importance of each objective. Such solution depends on our subjective preference for each objective.

The selection or ordering of the rules may then require a utility measure. For example, White and Sykes [200] proposed an interactive procedure which elicits the user's preferences to decide which rule in the set of alternatives has a higher priority. This procedure, based on Keeney and Raiffa's models of utility theory, produces a total ordering based on the weighted utility of each rule.

Notice, however, that despite of the inference-independent nature of the knowledge base, the designer (programmer,

or knowledge engineer) has to start dealing with the inner workings of the inference engine to obtain the desired behavior.

Besides inference control, we may need procedural attachments to (a) perform data retrieval from a database, (b) update a database, and (c) execute procedural programs.

Without loss of generality, procedural knowledge in rule-based systems, is possible by considering a predicate as a call to a procedure. The important point is to maintain the declarative nature of the knowledge base. In the inference engine developed in this research, this last approach was used to make a conclusion and to access standard computing facilities (e.g., a call to a search algorithm).

For example, suppose CM acknowledges a request for a cart at a location X. We may have the following rule in textual form:

```
if    (a cart is requested at location X)
then  (cart selected is nearest-cart to X)
```

Where "nearest-cart" is a function to obtain the nearest cart to the location X by using a shortest path algorithm (possibly a cart that is idle or it is about to be idle).

As seen along this section, we can use production rules to represent knowledge expressed by a "condition-action" pair. Antecedents and consequents, however, have been assumed to be either true or false (non-existing). Rule-

based representation can be also extended to those cases in which facts are given, and conclusions are drawn, with degrees of certainty, belief, or both. In particular, we will review probabilistic reasoning, where probabilities are associated with the facts and conclusions, and another representation method called "approximate reasoning."

Finally, we will briefly discuss two methods derived from research in natural language processing, namely, semantic networks, and frames and scripts.

2.4.3 Probabilistic Reasoning

Probabilistic reasoning applies when an object must be classified based on facts that cannot be simply stated as true or false, as two-valued logic does. Instead, the inference leads to likely outcomes with a certainty factor measured by a probability or some equivalent measure.

One approach to statistical classification is based on Bayesian statistical inference. First, the ES assumes that the object to be classified may belong to one of several mutually exclusive classes $C_1, C_2, C_3, \dots, C_n$ whose prior probabilities of occurrence are known. Second, the ES guides the user to conduct domain specific sampling or testing procedures yielding a number of patterns (features, or attributes) which are accounted by an event E . Next, the ES obtains, by calculation or elicitation, the conditional probability of a class C_i given the event E (i.e., how often

a class C_i is associated with the event E). Finally, the posterior probability that the object belongs to a class C_i given the patterns accounted by the event E , is determined by using Bayes' theorem.

The typical form of a rule is as follows:

if fact F_1 is present with evidence P_1 and fact F_2 is present with evidence $P_2...$, and so forth, then the conclusion is C with evidence Q .

Probabilistic reasoning is most common in areas such as medical diagnosis [148] and prospective mining. It is also applicable to decision problems involving risk analysis.

2.4.4 Approximate Reasoning

In approximate reasoning the problem is to infer from objects whose attributes are sets with ill-defined boundaries. A typical example is the use of linguistic variables. For instance, a person's income may be defined as low, average, or high; a project may have a profitability which is unattractive, or attractive.

In these examples, a particular outcome within an attribute is mapped onto a linguistic scale. This requires an association measure. When the boundary of a set cannot be defined with precision, the set is said to be fuzzy. Conventional sets with well-defined boundary are crisp.

In the standard mathematical notion of a set, an element either belongs to the set or does not (i.e., the membership relation is dichotomic). Fuzzy sets theory extends this restricted (perhaps unnatural) definition to account for those elements which partially qualify as members of the set and also as members of the complement. The degree of association of an element is measured by a degree of membership.

The degree of membership of an element of a fuzzy set is determined by a membership function bounded between zero and one. The elements (in the discrete case) or range of values (continuous case) that totally match the definition of the set, in a subjective manner, have a maximum membership equal to one. As the elements have features that makes them less representative than the prototype, the membership decreases to zero. Thus, the shape of this function can model the user's degree of belief of how well an element of the set fits the desired set definition.

Fuzzy sets allow mathematical manipulation of linguistic ordinal scales. For example, the scale: poor, average, good, excellent. Linguistic operators such as very, quite, or extremely, can also be handled by applying an algebraic transformation to the original membership function of the set.

For instance, in an investment evaluation problem, if the fuzzy set of 'good profits' has a membership function $f(x)$ asymptotically increasing between a minimum point $(X_0, 0)$ to

a maximum point $(X_1, 1)$, the set of 'very good profits' may have a membership function $g(x)$ equal to a power of $f(x)$. Given a 'good value' X for the profit, the concept 'very good' will always have a smaller or equal membership. In the example, both are equal to one at the maximum X .

In ESs that model approximate reasoning, the facts are presented with a degree of truth measured by the degree of membership between zero and one. The degree of truth of the inference is handled by fuzzy algebra [131,146].

Two basic results are that (a) the membership of the intersection of two fuzzy sets is the minimum of the membership of each set, and (b) the membership of the union is the maximum. Translated into fuzzy rules this will be as follows:

1. if fact F_1 is true at degree M_1 and fact F_2 is true at degree M_2 and fact F_n is true at degree M_n then the conclusion C is true at degree $\text{Min } (M_1, \dots, M_n)$.
2. if fact F_1 is true at degree M_1 or fact F_2 is true at degree M_2 or fact F_n is true at degree M_n then the conclusion C is true at degree $\text{Max } (M_1, \dots, M_n)$.

In addition, not only the antecedents of the rule may be approximate but also the consequent may include a degree of truth (less than, or equal to one). Such number multiplies the degree of truth of the conclusion.

2.4.5 Semantic Networks, Frames and Scripts

Semantic networks are based on a cognitive theory which postulates that humans store knowledge about objects by their attributes and links or relationships with other objects [10,p.81]. Put on a graph, the nodes or "units" represent the objects and the arcs or "slots" represent the attributes or relationships linking the objects.

Among the attributes, there may be relations expressing hierarchical dependencies, such as membership and inheritance. Such relations allow the deduction of new facts from the properties of the class to which an object belongs. Or, from the attributes of another object from which an entity descends. Thus, only the properties that make an object different from the others in its class need to be stored in the units.

Frames are based on psychological evidence (Minsky [126]) that humans deduce properties or identify objects based on prototypes stored in memory. These generic objects become distinguishable from others by learning which attributes the object (or scene) should exhibit. Thus, a frame is a prototype filled out with assumptions or "expectations" about the characteristics of the object. Some of these properties are default assumptions which are eventually changed as new evidence arises in the inference process.

Scripts are frame-like representations for a sequence of generic events in a scene (e.g., the steps followed by a person visiting a clinic).

A common inference method used in frames starts with an incomplete set of attributes. Several hypotheses to determine the classification of the object are formulated. Each hypothesis is tested by accessing known attributes in the frame. If not available, the attributes may be obtained from other frames the object is related to (by property inheritance or membership). This also involves default assumptions, invoking procedures, and asking users about unknown properties. Since the retrieval of new information may discard one or more of the initial hypotheses, new ones may appear. The inference is then a cyclic hypothesize-and-test procedure until a satisfactory explanation is derived [148, 204].

In frames, procedural attachments can be implemented by using so-called demons. Demons are procedures activated whenever a particular task is required. Their use includes retrieval of properties by using formulas or running other programs, prompts to the user to obtain missing values, pointing to other frames, and maintenance activities such as updates, deletions, and creation of frames.

Frames, as well as all the methods seen above, require that the key properties defining the objects, are predetermined. Thus, the efficiency and effectiveness of the

programs depends on how complete such characterization can be done. This, in turn, is a function of how well the problem domain is defined.

A common, often unexplained condition found in ES construction is that the domain is "small and well-defined." Clearly, in a "large and ill-defined" domain, the properties that completely describe the objects become more difficult to find and the possibility of finding unknown or exception situations increases. An ES simply fails when it does not have enough knowledge to conclude something.

The effectiveness of an ES decreases as the number of properties to be considered increases. Unfortunately, there are no rules or methods to assess the suitability of the domain before developing an ES. Much depends on the knowledge engineer's intuition.

At some extent, this is an interpretation of the phenomena of blindness in the critique of AI by Winograd and Flores [203]. The authors argue that the very condition of knowing the properties of the objects, precludes AI from reaching performance comparable to natural intelligence. Our own notion of intelligence relates to our ability to do reasoning even when we do not know the properties nor the mechanics to generate and systematically search the space of solutions.

2.5 Selection of a KR Method

We have seen that KR has direct connection with the form in which knowledge about the domain is available. Also, it is related to the type of attributes of the objects involved.

Some researchers have attempted simple rules to suggest what method is more appropriate for a particular problem. For instance, Ramsey, Reggia, and Nau [148] have conducted experiments in medical diagnosis. They have formulated the same problem under several KR methods and assessed the performance in each case. Performance deals with the power of the method to conclude correctly, and the ease of use and learning from the user's viewpoint.

Although the authors' findings derive from a medical domain, their results can be generalized to other applications. Three factors are important in selecting KR:

1. Form in which the available knowledge is organized: whenever possible, it is desirable that the KR method resembles the form in which the input knowledge is available. Forcing conversion to a specific representation may not only be expensive but also can lead to modeling errors. (For instance, in a decision process it may be difficult to express a rule when the cause-effect relationship is not clear.) Input knowledge can be available as (a) branching logic or decision trees, (b) decision rules of the if-then type, (c) decision tables,

(d) in descriptive form as in books, articles, manuals, or tabulations.

2. Degree of uncertainty or fuzziness in the attributes of the objects: a KR method should consider the uncertainty due to randomness or fuzziness. Knowledge may be categorical, facts and conclusions are stated with absolute certainty; it may be uncertain, but the probabilities (as in Bayes' theorem) must be available or obtainable with accuracy; it may be approximate, but the degrees of truth (as in fuzzy sets) must model correctly the perceptions of the knowledge source and the user. For example, if an expert gives the rule "if the improvement in quality is high then install the machine," both the expert and the user should be referring to the concept "high improvement in quality" in the same scale of preferences. The certainty of the knowledge also determines the type of inference that is possible. Thus, in rule-based systems we can obtain deductions that are mutually exclusive with respect to some of the object's attributes but it would be inappropriate for those cases when several outcomes can result, as in probabilistic inference.

3. Degree of context dependency of the inference: this deals with the information needed to deduce a particular fact. In rule-based KR, when too many antecedents, or a complex interaction with other rules are needed to

conclude a fact, the system may lose its clarity and efficiency. We would have to decompose a rule into many sub-rules making the inference more complex. In predicate calculus a formula with a large number of and/or atomic formulas can be decomposed into various sub-formulas at the cost of a larger knowledge base.

Based on the factors above, we have the following rules:

Rule 1. If knowledge preexists in descriptive format such as books and articles, and there is a mixture of probabilistic and categorical classification, and there is a high context dependency and multiple potential outcomes, then frames are more appropriate than other methods.

Rule 2. If knowledge is organized in rules or decision tables and the classification is predominantly categorical and there is little context dependency, then rule-based deduction is appropriate.

Rule 3. If probabilities are obtainable, and the assumptions of Bayes' theorem hold, then statistical pattern classification is appropriate.

Rule 4. If a linguistic scale is to be used, or the attributes of the objects are fuzzy, then approximate reasoning is appropriate.

2.5.1 Tool Selection

The choice of the tool to develop a knowledge-based system mainly depends on the KR method that is correct for the problem under investigation. We do not discuss the tools here. In recent years a number of publications have appeared describing different programs and therefore only a brief reference to the sources is presented.

The predominant language for AI research has been LISP [204] and more recently PROLOG [30]. LISP-based tools vary from the basic primitives of the language to complete hardware (e.g., LISP-based machines) and software environments (shells) supporting some of the KR methods discussed in 2.4. Classic examples of these environments running on mainframes are: OPS5, EMYCIN, ROSIE, KAS, AGE, KEE, ART, Knowledge Craft, IBM's Expert Systems Environment. A number of other tools for microcomputers include: Texas Instruments Personal Consultant, various PROLOG and LISP-based environments, Expert Ease, Xi-Plus, EXSYS.

An excellent survey on tools is given in Harmon and King [78]. Classical books dealing with early applications and shells (before 1982) have been written by Barr and Feigenbaum [10], and Hayes-Roth, Waterman and Lenat [82]. Newer applications in general areas can be found in a paper by Buchanan [32]. Specific commercial products are reviewed in [41,65,66,183].

Presently, many commercial tools are available. From an end-user's viewpoint the investment in a large-scale ES development system may even justify multi-attribute techniques. That is, a set of problem-specific attributes, monetary and non-monetary [152], are associated with the different candidate tools. Then, a multi-attribute ranking technique is applied to select one. An example of this approach is found in Liebowitz [114] who used the Analytical Hierarchy Process (AHP) and made selections consistent with the end-user application.

In Chapter 3 we review the literature on AI as applied to analysis, planning, and control in manufacturing systems. The KR methods used in these works fall into the categories discussed in preceding sections of this chapter.

CHAPTER 3

LITERATURE SURVEY ON AI FOR ANALYSIS, PRODUCTION
PLANNING, AND CONTROL IN MANUFACTURING SYSTEMS

In this chapter we review the status of AI research in manufacturing systems and situate the present research in perspective. The works considered in this survey belong or are closely related to the manufacturing domain. In particular, we consider the managerial aspects of computer integrated manufacturing (CIM) systems. Since the concept of CIM also applies to computer aided decision support systems, we briefly examine the role of AI in the fields of modeling, and economic evaluation of these systems. Finally, we point out areas for further investigation in AI for production planning and control, and sketch the methodology of the research.

3.1 Industrial Applications of Knowledge-based Systems

In industry, ESSs have been proposed in the following areas:

1. Evaluation of investments in computer integrated manufacturing (CIM) systems.
2. Monitoring, control, and emergency management of power plants, computer complexes, and oil industry.
3. Fault diagnosis and repair of complex machinery and electronic equipment.

4. Control and operations scheduling in manufacturing systems.

Rule-based ESs have been used to justify projects in CIM systems (LeClair and Sullivan [111], Sullivan and LeClair [177]). The problem solving method classifies a project into three basic categories: execution, no execution, or postponement. The ES, called XVENTURE, reaches one of the goals by eliciting information about a number of non-monetary attributes (e.g., improvement in the firm's competitive position, quality of the products, reduction in scrap material, and flexibility). XVENTURE uses the non-monetary indicators in addition to internal rate of return and present worth. XVENTURE is built on a multi-expert architecture because the advice of multiple experts in the field has not reached consensus (LeClair [110]).

Although most of the non-monetary attributes are valued in linguistic scales (e.g., using qualifiers as little, average, or significant improvement in productivity), the rules are categorical. Consequently, a "small" variation in the linguistic qualifier of one or more attributes can shift the final conclusion without considering intermediate levels or a belief measure. In this case, the use of a linguistic scale implicitly assumes that the user's perception about the merit of an attribute (e.g., "significant improvement in productivity") is the same as the expert's.

Further research in this class of problems can be done by using approximate reasoning methods, as described in section 2.4.4. Conversely, ESs can incorporate into their knowledge bases and inference engines the broad background on multi-attribute, group decision-making provided by OR.

Some research has been done in this area. For instance, Borisov and Naglis [27] designed an ES embedded in a computer aided design (CAD) system to evaluate options in robotic installations. The system works under multi-criteria decision-making and the ordering of alternatives is obtained by using linguistic scales and fuzzy sets. In section 2.4.2 we discussed the work of White and Sykes [200], who applied utility theory to resolve conflicts among various rules leading to different advice.

ESs have also been developed for assisting operators at electric power plants (Talukdar et al. [186]), computer complexes (Milliken et al. [125]), and oil industry (Sachs et al. [158]). In these installations, data collection and status report capabilities are built into the software and hardware but the simultaneous control of critical variables is often performed by a human operator.

Of special importance is the management of emergencies. A process variable goes outside the safety range and the operator must quickly execute appropriate actions. Since some abnormal events may seldom occur, the operator would

require additional information to handle the emergency. The steps to bring the system back to its normal operations are normally documented in manuals. Nevertheless, when time is critical, there may not be enough time to sift through the text to find the remedial action and there is a potential for large errors.

A goal for control ESs is to perform corrective actions in unattended mode. So far, the literature has not reported fully autonomous systems due to limitations in assessing all possible abnormal conditions and the risk involved in having an inconclusive system.

A control ES continuously monitors system variables. Whenever control recognizes a combination of events, the program diagnoses the situation and suggests the sequence of proper actions. Hayes-Roth, Waterman, and Lenat [82, pp.13-15] characterized this as a data-driven recognition-diagnosis-action cycle.

Many diagnosis problems can be thought as a classification problems with the classes being the goals of the search (e.g., types of malfunction, identification of a polluting substance). In these cases, backward-chaining inference is an adequate model.

When the set of goals cannot be enumerated in advance and only a set of preliminary facts is present, forward-chaining inference is appropriate.

Another area for ES application is fault diagnosis and repair of machinery and electronic equipment. A technician assisted by an ES isolates the cause of malfunction of the equipment by conducting several tests on different components. The ES guides the technician in performing the tests, usually aided by computer graphics or TV displays, and suggests the corrective actions.

ESs in this domain have been constructed by Fink et al. [59], Folley and Hritz [63], Hofman et al. [87]. Herrod [184] reports other applications for troubleshooting in communication networks and equipment for the semiconductor, food, and glass industry. Pau [140] presents a survey on ESs for diagnosis and repair with an in-depth analysis of their architectures.

Among the benefits of using these systems, we can cite (a) shorter time to diagnose and perform the repair task, (b) training of maintenance personnel, and (c) lesser requirements for scarce, skillful technicians.

3.2 Meta-Modeling

Knowledge engineering concepts, such as declarative knowledge representation, domain-independence of the inference engine, and explanation of reasoning, are being applied in the modeling domain. Research in this area intends to facilitate the construction and use of mathematical and simulation models by non-analyst users. It also

attempts to design self-organizing models building on primitives such as basic models and algorithms.

We call this research meta-modeling because the AI programs intends to model the reasoning of the analyst who formulates the models. Two areas of investigation are (a) the selection and use of models for optimization and forecasting, and (b) knowledge-based simulation. The latter deals with formulation and analysis of simulation models.

3.2.1 Selection and Use of Models

A problem some decision makers must face, is the selection of a correct model and efficient algorithm to solve a given problem. The knowledge base consists of combinations of possible formulations and solution techniques related to the problem domain. The characteristics of the problem may be ill-defined (or stated in vague terms) and it is difficult for the user to determine whether the assumptions of the models fit the problem description. In addition, such user may be unfamiliar with the modeling terminology and theoretical foundations.

The knowledge-based approach is therefore used to elicit the characteristics of the problem, explain the modeling concepts involved, and determine the model that adapts best to the problem description. For instance, Rogers and Barthelemy [154] describe one these systems to assist engineers in choosing models for mechanical design. Weitz

[199] built NOSTRADAMUS, an expert system to assist non-statisticians in selecting forecasting methods. NOSTRADAMUS was built with the EMYCIN shell (the inference engine of MYCIN, a program for medical diagnosis) and supports certainty factors in the rules.

As indicated earlier in Chapter 1, one problem that limits the creativity of model builders is the lack of standardized AI tools to access, or interact with, many efficient tools already developed in specialized domains (e.g., statistics and mathematical programming).

In the case study by Weitz [199], NOSTRADAMUS did not perform computations on the data set. It had to ask the user an initial clue about the data pattern by simple visual inspection. Since the answer was, in general, non-categorical, the user could respond by using certainty factors. This approach led to two problems. First, the program had to obtain statistical measures in technical terminology, in contrast to its initial purpose which was to avoid technical jargon. Second, the subjective certainty factors led to conceptual errors and caused the expert system to make wrong or unclear conclusions. As a result, the final diagnosis advised, for instance, using Box-Jenkins with a certainty factor of, say, 0.6. The certainty factor was unrelated to the correlation of the model. And, it was possible that an alternative model previously rejected by its low certainty factor (as a result of an inappropriate input from the user)

had a better overall response. If the shell had been able to access a statistical module, the analytical work could have been done independently of the user.

In the discussion above, models and algorithms are given. The problem is to select the correct one. The inverse process, constructing a model from dialogue with the user, is much more difficult and it has not yet been extensively investigated. A few papers, mainly in the area of simulation, have addressed this problem.

3.2.2 Knowledge-based Simulation

Knowledge-based simulation intends to facilitate the formulation, interpretation, and maintenance of simulation models.

In low-level tasks, discrete event simulation heavily relies on list processing; a queue can be represented as a list, events are recorded on a sorted list, and handling queue priorities requires list sorting. AI related languages (in particular LISP, and at some extent, PROLOG) predominate over other languages in these tasks.

Several authors have written interpreters and translators in LISP and PROLOG to simplify simulation programming. See for example, Adelsberger and Neumann [1], Cleary et al. [42], Elmaghraby and Jagannathan [52], Futo [70]. Ruiz-Mayer, Talavage, and Ben-Arieh [155] present preliminary results on SYMION, an AI programming environment for simula-

tion of networks. De Swaan Arons [49] and O'Keefe [135] discuss examples integrating between AI and simulation.

In more AI-oriented research, knowledge-based simulation deals with model formulation from a narrative description and parameter optimization:

1. Formulation from narrative description. Natural language processing has been applied in the construction of queuing models. The natural language processor captures the user's informal description of a queuing situation (e.g., in English) and produces a model which is later processed by a conventional simulation package. These systems are built around a natural language parser and a lexical analyzer to identify the actors, elements, and flows of the system. It also needs a syntactic analyzer to check the consistency of the dialogue and an interpreter to generate the lower-level code in a simulation language or a conventional programming language. Heindor (in Barr and Feigenbaum [10, Vol.3]) developed in 1972 the NLPQ system (Natural Language Programming for Queing Simulations). The system parsed sentences and generated GPSS code by using production rules. More recent research have been conducted by Paul and Doukidis [141]. Current developments, however, are still limited to very simple queuing situations such as the single-server model.

2. Parameter optimization. Frame-based knowledge representation has been used to analyze the results from various simulation runs under different scenarios. The program looks for causal relationships among specified variables and suggests directions in which the variables can be optimized. Reddy et al. [150], describe a model to evaluate distribution plans over a network of vendors, customers and warehousing facilities. The attributes of the entities and their temporal and causal relationship are expressed in schemas or frames (as described in section 2.4.5). The model was developed in the Schema Representation Language (SRL). As reported by the authors, the system runs the simulation, interprets the results, and automatically tunes selected parameters for optimization.

3.3 Knowledge-based Systems in Planning and Control

AI research for scheduling and control of automated manufacturing systems began to appear in the literature by 1980. The creation of several flexible manufacturing systems by the end of the previous decade, had given origin to new requirements in planning and control systems. One of these, was the design of computer-based systems that could work in real-time.

Next, we point out the main arguments in favor of knowledge-based systems. We use the term "cell manager" to

refer to the person in charge of the shop floor operations. Such person performs the function of a supervisor, dispatcher, or scheduler.

1. The decision process involved in the control of a manufacturing facility is knowledge intensive. Managing machines, robots, transporters, fixtures, pallets, materials, human resources, and computer operations, requires a highly skilled cell manager. The manager's expertise may be undocumented, partially retrievable from manuals, changing through time, or perishable due to retirement.

2. Although the commands issued by the operator can be directly executed by the control software, the managerial decisions that affect the system's performance are not automated. The operator's knowledge must be encoded in the control software.

3. The cell manager faces the paradigm of excess of variety which is common in cybernetic systems (Ashby [8]). That is, the system can be in too many states for a human to control them effectively and in real-time. In other words, time constraints and information overload may cause errors, delays, and low efficiency.

4. Mathematical models for planning can lead to better policies than heuristics do. Nevertheless, these models are in practice unsolvable or inefficient because actual operations require too many time-dependent variables and planning is perturbed by random events. Even if computa-

ble models for real-time planning and control can be formulated, intervention to correct random disturbances is still necessary.

5. Automated manufacturing systems are built incrementally. Software design needs to be flexible enough to accept changes in the floor layout and operating rules. In knowledge engineering, these two components reside in the knowledge base. Thus, the control software can be developed in stages, without prior knowledge about the final application.

The remaining sections present a survey of important contributions to the field of scheduling and control under the AI approach.

3.4 Progress of AI Research in Scheduling and Control

In 1980, Bullers, Nof and Whinston [33] showed how AI could be used to control an automated manufacturing cell. At that time, AI was mainly confined to experimental work and automated reasoning systems were known by their origins in predicate calculus and automated theorem proving. Today, their work could be reformulated by using a logical programming language (e.g., PROLOG). The authors, however, did not elaborate on the knowledge needed for scheduling and control.

In 1983, Fox [67] developed the Intelligent Scheduling and Information System (ISIS) for production planning in machining shops. ISIS developed schedules at different stages of the production of a product. It generated and selected alternatives by using "constraint-directed reasoning." This heuristic method derives from constraints imposed by manufacturing engineers when choosing machines to route the parts. The tolerances in the final product depends on this routing. Thus, the purpose of ISIS was to generate acceptable sequences so tolerances in the final product were met.

More recently, Ben-Arieh [12,13] developed a knowledge-based simulation to address the dynamic assignment of work stations to assembly operations in a flexible manufacturing system. Upon completion of an operation on a given part, the next machine was selected by using heuristic search. The evaluation function consisted of a combined cost depending on queuing time and machine utilization. The criteria to measure the performance of the system included the production per machine, productivity per part type, average queue length, and machine utilization. The results showed that, in general, the heuristic method performed better than static sequencing and human experts. This was attributed to the use of real-time information on job and cell status, and the "look ahead" evaluation of alternative routings. Ben-Arieh

addressed FMS control, but the material handling system nor storage capacity in the cell were considered.

The simulation model was written in PROLOG and the heuristic search implemented in PASCAL. A difficulty arising in PROLOG clauses as a KR method is the use of meta-logical elements to control backtracking (the PROLOG engine). Using elements such as "cuts and fails," requires knowledge about PROLOG programming. The final program becomes, in general, procedural. A change in the cell layout or in the operational rules would require reprogramming the system. At some extent, this makes the model domain-specific and limits the expressiveness of KR. But, this could be corrected by building a parser in the same PROLOG language.

Bruno, Elia and Laface [31] also developed a hybrid system for scheduling in a FMS. The model combines a rule-based ES, written in OPS5, with a network of queues model. The system must schedule several lots over a period of two weeks. The lots are introduced into the FMS according to a priority indicator. But, due to the loading conditions, the performance of the FMS and the status of other lots already loaded is affected when the new lot enters the FMS. In order to measure the effect of the new lot, approximate performance indicators are obtained from the network of queues model. The rules of the ES examine these effects and test other alternative lots if they improve the performance. This system evolved from an existing simulation model

written in FORTRAN. In the new system, the authors indicate that no major decrease in computational efficiency occurs. But, major benefits arise from the maintainability viewpoint since the knowledge representation method allows them to encode rather complicated heuristic rules. This would have been cumbersome if done under procedural representation.

Lawrence and Morton [109] describe a developing system, called PATRIARCH. Here, production scheduling is defined in a much broader sense compared to short term loading and routing decisions. It involves strategic planning at the firm level, master schedule at the plant level, loading at the work centers, and dispatch schedules at the work station level.

PATRIARCH uses Net Present Worth (NPW) as a measure of effectiveness to overcome possible conflicts in other measures such as makespan and tardiness. It is assumed that maximizing the NPW would encompass most of the traditional scheduling objectives. The authors report that AI knowledge representation techniques and heuristics are used. However, no further insight into the system architecture or progress is provided.

Chiodini [39] describes SCORE (Shop floor Contingency Rescheduling Expert). SCORE is also a developing prototype for real-time scheduling in a Just in Time (JIT) manufacturing system. The basic functions of SCORE are to (a) schedule assembly lines with the constraints received from the master

production schedule, (b) predict the propagation of abnormal events in the assembly lines, (c) re-schedule the lines in case of abnormal events, and (d) coordinate the schedule of product components with the end-product assembly operations. This system uses rule-based and frame representation and it is being developed by using the Knowledge Craft shell.

Herrod [84], reports an AI system programmed in LISP for planning, scheduling, and dispatching in a flexible manufacturing system at Texas Instruments Inc. It consists of an expert system to encode the system manager's existing manual practices.

Steffen [176], presented a survey on AI and ESs for scheduling. He observed that despite the number of publications describing applications, only two systems can be considered operational: ISA and DISPATCHER.

ISA and DISPATCHER belong to a series of ESs (XSEL, XCON, XSITE, ISA, INET, ILOG, DISPATCHER) developed by Digital Equipment Corp. to carry out the entire manufacturing process of customized computer configurations [50]. When a customer originates an order, XSEL translates the request into a company order according to functional specifications. XCON converts the order into a bill of materials and a configuration diagram. XSITE uses the output from XCON to determine the equipment layout. ISA determines the schedule of the order. In production, ILOG is used to resolve conflicts that arise from underestimated delivery dates, mate-

rial shortages and random delays. INET is used to route the order through the factory, warehouse, and distribution centers. DISPATCHER monitors and control automated material handling devices.

The economic benefits of using ISA and DISPATCHER were difficult to estimate before their implementation. Steffen indicates that the firm reduced order processing time from 10-15 minutes to 10 seconds, and the production rates increased between 50 and 100 percent.

The fundamental ideas on AI for production scheduling and control have been drawn along this chapter. Supplementary discussions and examples can be found in Ford and Floyd [64], Grant [74], Kusiak [108], Robbins [153], Ulicny et al. [190], Wysk et al. [206], Young and Rossi [211].

3.5 Development Environments for Manufacturing

Clearly, advances in AI for planning and control of automated manufacturing would be difficult to achieve without an appropriate software development environment. As discussed earlier in section 1.1, even for those systems that do not require direct interaction with the shop floor devices, substantial work have had to be done in order to interface them with software packages and databases.

Specialized environments are still scarce and in practice only one of these is operational. In 1984, Wright and Bourne

[205] created the Cell Management Language (CML) to address the design and control of automated cells.

CML (now commercialized by Westinghouse) possesses several characteristics worth to consider. For example, an operator without major background in computer programming can design a sequence of commands so a robot performs a given task. The operator builds the robot program by telling CML what actions (consequents) would follow when a combination of facts (antecedents) are present. This is assisted by a rule-based language. A complete program usually requires several passes since not all the facts required to trigger an action are evident at the beginning. Thus, a full control system is built incrementally.

CML also has utilities for handling different communication protocols and allows object-oriented programming. The latter is a programming style more evolved than programming with libraries of functions and subroutines. Programs and data become objects (usually represented by icons in a graphical user interface) which intercommunicate by messages rather than by parameter-passing. Large programs can be constructed quickly out of modular objects which could be developed separately.

The book by Wright and Bourne is a recommended reading to learn about various AI efforts (in robotics, vision, and knowledge-based systems) to fully achieve manufacturing automation. They have coined the term "manufacturing intel-

ligence" to refer to the production of small batches of parts without human intervention. Their emphasis, however, is on AI applications intended to replace human machinists. Managerial aspects of manufacturing are addressed superficially.

In the near future, research in manufacturing intelligence will probably concentrate in the design of inexpensive firmware acting directly onto the low-level equipment controllers. An application in this area is described in Lebow and Blankenship [113] (U. of Maryland), who designed a knowledge-based system to drive a programmable logic controller (PLC) in a continuous process. The system tunes the parameters of the process, in real-time, by using a Proportional-Integral-Derivative (PID) control model. The inference is based on fuzzy logic. A prototype have been tested by simulating the process and the results have shown its viability. The final system is intended to run on an 80186-based microcomputer under DOS with a TI LISP coprocessor.

3.6 Contributions of This Research

Going back to the issue of real-time management of operations in a CIM system, we note that this includes two phases. First, there is a planning function which develops a schedule. Second, there is a control function which, in real-time, must handle the loading of parts, choosing the

routes to be followed, assigning resources, monitoring, and handling random events such as equipment failures. Both phases are integrated. Control feeds information back to the planning phase to adapt the schedule to present conditions of the shop floor.

According to Gershwin [73], the common approach used in the two phases above is that of "dispatch scheduling." Decisions are made on "as-needed" basis without planning. For instance, in the "scheduling software" for Kearney & Trecker FMSs [96] it is stated that the scheduler "does not base its decisions on projections of the future."

Both, planning and real-time control, require an analysis of their effects: on the system as a whole and over time. For this reason, we cannot always rely on emulating a human expert. The analysis of such effects requires information provided by a model of the system.

By following the model-based approach, Ben-Arieh [12] developed a knowledge-based simulation and evaluated alternative routings by using heuristic search. Bruno et al. [31] evaluated schedules with a network of queues model.

Notice, however, that these two approaches do not provide adaptive scheduling and control. The study by Bruno et al., is oriented to planning but it is not intended to work in real-time. Ben-Arieh's system chooses machines as the parts move within the cell but it does not plan for capacity losses due to corrective maintenance.

Building on the works of Ben-Arieh, Bruno et al., and the ideas contained in CML, this research intends to find what model-based knowledge is required to perform real-time control and adaptive scheduling.

For this purpose, we consider (a) an adaptive planning task which determines a good trajectory for the production variables and (b) a control function which monitors the behavior of these variables. The control function corrects deviations from the target values and manages all the lower level tasks of dispatching and routing by means of a knowledge-based system.

The manufacturing system is dynamic. Therefore, a tool is designed to encode the knowledge and evaluate how the system evolves as the decision rules are applied over time.

CHAPTER 4

PRODUCTION PLANNING AND CONTROL IN COMPUTER INTEGRATED
MANUFACTURING SYSTEMS

Computer Integrated Manufacturing (CIM) comprises several managerial and operational functions which are performed by, or with the assistance of computers. Among these functions is Production Planning and Control (PPC). PPC is responsible for production scheduling, resource assignment, and achievement of the plans.

The objective of this chapter is to examine the functions of CIM stressing on PPC. In particular, we emphasize the use of a hierarchical schema to perform distributed planning and control.

We conclude the chapter by discussing further the approach of this research to solve the problem defined in Chapter 1.

4.1 CIM, Production Planning, and Control

The loss of competitiveness of the U.S. manufacturing industry in the sector of high-variety small batch manufacturing, and the consequent need for revitalization, originated the concept of CIM [93,111,129,182,191].

A computer integrated factory [129] is envisioned as a system in which the data generated from computer aided design (CAD) feeds the production planning function of

computer aided manufacturing (CAM) (Figure 4.1). Planning selects proper equipment, processes, sequence of operations, and operating conditions for manufacturing. Planning passes the information to the control function which directs the equipment and machines to produce parts and assemble the products. Computer-controlled machines and equipment can automatically adjust their operating conditions, select tooling, handle parts locally, execute the process, and perform quality control. To close the loop, there exists feedback between the equipment and the control function, and between control and planning.

Thus, CIM deals with the integration of CAD and the three phases of CAM: planning, control, and manufacturing of a product.

Since 1970, CIM has experimented major progress in CAD and the operational phase of CAM through the use of flexible manufacturing systems (FMSs).

A FMS is intended for mid-volume, medium-variety production (e.g., 15-15,000 parts and 2-800 varieties) with the economical effectiveness of a high-volume transfer line [96]. This is accomplished by a group of flexible programmable machine tools and material handling systems (MHSs). A MHS usually consists of robots, automated guided vehicles (AGVs), and automated storage and retrieval (AS/AR) systems. The FMS is controlled by a central computer or in distributed form by a hierarchical network of computers.

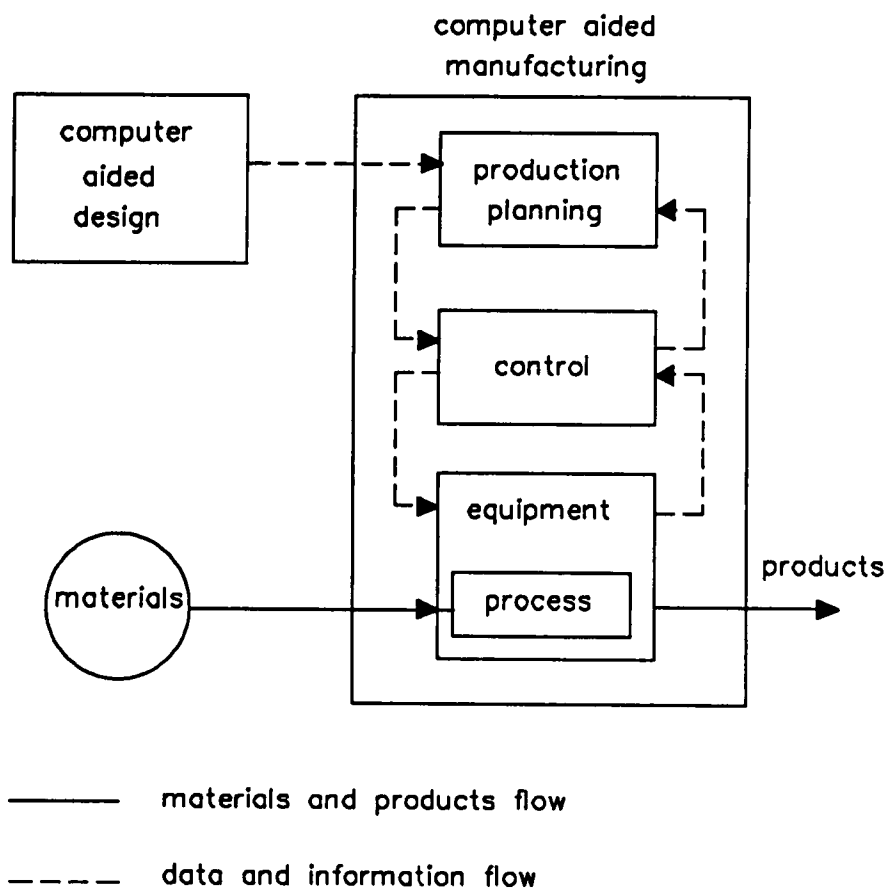


Figure 4.1: Main Functions of CIM Systems.

Although the definition of CIM can also be extended to the process industry and other types of automated manufacturing such as transfer lines, we focus on discrete parts manufacturing and FMSs.

In PPC, CIM still relies on methods utilized in conventional systems. Information systems assist operators and managers who then make the decisions with tools inherited from traditional manufacturing. For instance, in section 3.6 we noted that shop floor dispatching is mostly done on "as needed" basis. More elaborated approaches use priority scheduling but without planning capacity or reacting to random events such as machine or equipment failures. We will discuss methods for different levels of scheduling in section 4.3.

The benefits of providing computers with greater decision making capability in planning and control decisions are analogous to benefits gained by CAD, namely, improved speed, less errors, increased depth of analysis in less time, and increased personnel's productivity.

4.2 Schema for Planning and Control

In order to examine the interactions between managerial and operational functions performed in a CIM system, we use a hierarchical decomposition. Mesarovic, Macko, and Takahara [124] characterized a hierarchical system as a multi-layer system in which long-term complex goals are decomposed into

short-term simple tasks to be handled by subordinate units. The units have a degree of autonomy determined by the scope of their assigned tasks. Information and data may flow between any pair of components at, and within, any layer. But, feedback is mainly between adjacent layers.

PPC functions belong to a hierarchical decision system since they are performed at various levels of the manufacturing system, with increasing level of detail downwards, and over shorter planning horizons. This approach also simplifies the analysis for modeling purposes. Otherwise, global planning and control would be extremely complex to achieve.

As shown in Figure 4.2, a CIM system consists of five layers: facility, shops, cells, workstations, and equipment (processes). Each layer may contain several subsystems. Each subsystem performs its own planning and control functions.

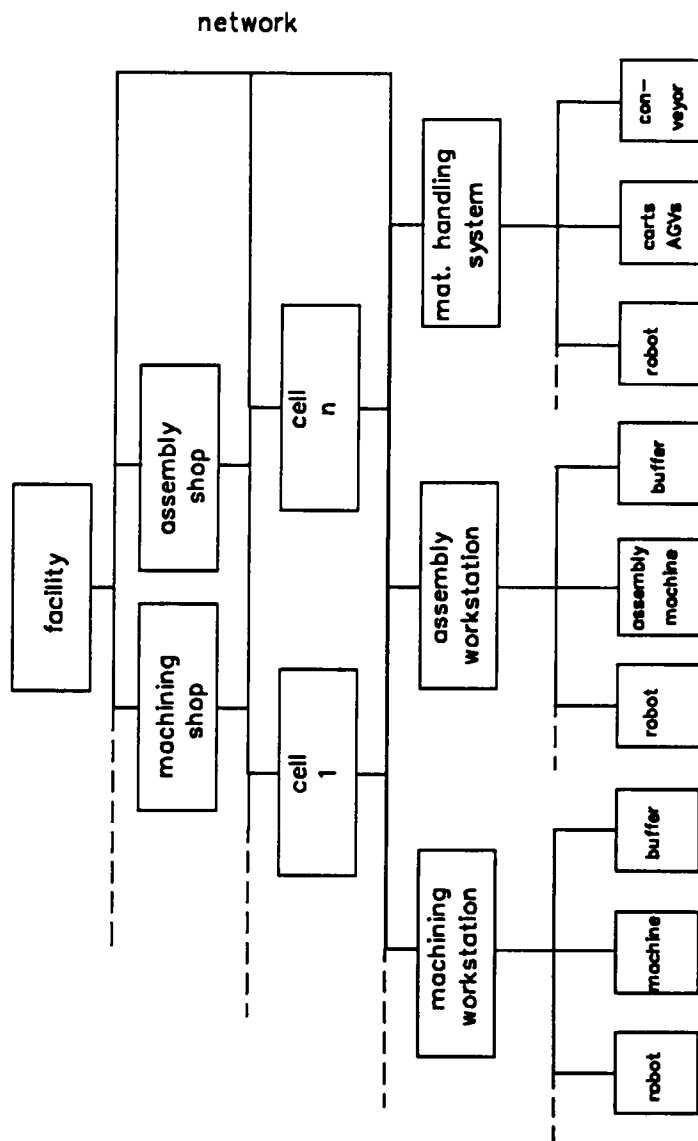
We might add on top the enterprise layer which determines the long-term goals for the whole organization. For our abstraction level, however, we restrict the analysis to the five layers.

Facility. The facility layer represents the factory and it performs the following functions:

(a) Information management: handles the information systems to support inventory management, cost accounting, customer order processing, and purchasing.

LAYER

1	facility
2	shop
3	cell
4	work station
5	equipment



Source: Adapted from Jones and McLean [95].

Figure 4.2: Hierarchical Architecture of CIM Systems.

(b) Manufacturing engineering: is concerned with computer aided design (CAD), analysis of parts and fixtures, process planning, and programming and maintenance of numerical control (NC) programs.

(c) Production planning: determines long term, company-wide production schedules, perform project management, and long term capacity planning.

Shop. The shop is a physical grouping of workstations for machining, measuring, and assembly operations, and material handling devices for storing and moving parts. Its functions are:

(a) Batching: classifies job orders into batches of similar parts and determines the size of the batches.

(b) Task scheduling: performs short term scheduling of batch orders and equipment maintenance.

(c) Resource allocation: assigns workstations, storage buffers, tools, and materials to the scheduled batch jobs.

Cells. Cells consist of workstations interconnected, not necessarily contiguous, under computer control. A virtual cell (Jones and McLean [95]) is a logical arrangement of workstations for the production of a batch (as if several workers with different skills formed a work team for a particular job). It may be reconfigured if the process plan

for another batch requires a different logical layout. Typical FMSs consist of cells with fixed configuration. In cellular manufacturing each cell is formed by contiguous family-oriented workstations served by NC machines and MH devices (e.g. Group Technology cells [77]). The functions at the cell layer include:

(a) Dispatching and routing: human or computerized controllers perform real-time dispatching and routing, manage workstation sharing among several batches (unit batches), and request resources such as transportation devices, pallets, and fixtures.

(b) Monitoring: controllers check resource availability, job progress, machine status, and transporter status.

Workstations. The workstation controller issues commands to the material handling equipment and machines to setup the jobs, execute the process, and remove them from the workstation.

Equipment. Programmable logic controllers (PLCs) are finally given the task of making the floor shop devices execute their functions. Computer controlled (NC, DNC) machines process materials (e.g., metal cutting) or assemble parts (e.g., insertion of chips on boards); automated measuring devices perform local quality control for the

parts being produced, adjust machine precision and operating conditions; change and replace tools; robots move work pieces from and to the machines and transporters; AS/AR systems move and hold materials, work in progress, and finished products; conveyors, and/or AGVs move parts between locations.

4.2.1 Manufacturing Database

The manufacturing database management system stores and maintains the data necessary to execute the PPC functions. It performs the following tasks (Young [211]):

1. Stores the system's historical data, such as production levels, productivity, equipment utilization, and machining times.
2. Maintains a library of NC programs which are downloaded onto direct numerical control (DNC) machines.
3. Stores the processing sequence of each part through the workstations.
4. Stores the operation-to-machine assignment table.
5. Stores the pallet-to-part assignment table.
6. Stores the tooling-to-machine assignment table.
7. Provides supports for database query and the user-interface to maintain and use all of the above.
8. Maintains a log of events occurring on the shop floor to allow recovery in case of a system failure.

9. Keeps real-time information on the status of parts in the cell, machines, MHS components, and pallets.

4.2.2 Hierarchical Integration

In an integrated computer-based planning and control system the following conditions must be met:

1. Each subsystem (Figure 4.2) must be provided with specific scope and goals.
2. Goals and data must be expressed in numerical or symbolic form. The data must reside in the databases of the system and it must be transferable between layers by a LAN.
3. Goals must be decomposable into tasks for the next lower layer until they become specific instructions to the equipment controllers.
4. The control feedback mechanism must assure each subsystem achieves its goals effectively.

In each subsystem there is a controller. The planning system determines the goals and the controller uses its knowledge to achieve them. At any layer, there exist knowledge-intensive tasks. These are more noticeable at the facility, shop, and cell layers where control relies on humans. At the workstations and equipment layers, control requires a mixture of both human and machine intelligence.

For instance, in section 3.6 we saw that even the work of

the machinist, traditionally considered as a mechanistic task, has proven to be very difficult when attempting automation.

In general, knowledge concerns heuristics, algorithms, or both. A controller uses its knowledge to:

1. Process control messages from the next higher layer.
2. Send control messages to one or more units at the next lower layer.
3. Report its status to the next higher layer.
4. Process feedback from the next lower layer(s).
5. Request sensory data from other controllers in the environment.
6. Send sensory data requested by other controllers.

4.2.3 Computer Control Architectures

Related to the architecture of PPC is the computer control architecture. That is, how computers and equipment controllers are logically arranged to transfer data among them and execute the manufacturing operations.

Since 1980, the National Bureau of Standards (NBS) has been making efforts to standardize the wide spectrum of control schemes devised by individual manufacturers of automated equipment. The Automated Manufacturing Research Facility (AMRF) of the NBS is built around a hierarchical architecture. As pointed out in several studies conducted at

the AMRF, this has been done to resemble the organization of the PPC functions, as described earlier (Albus et al. [7], Bloom and McLean [23], Furlani et al. [69], McLean, Mitchell, and Barkmeyer [121], Simpson, Hocken, and Albus [173]).

The AMRF has implemented hierarchical control in robots and other material handling systems at the cell layer (Scott and Strouss [161], Szabo, Juberts and Kilmer [184]).

In recent years, as CIM systems have become increasingly complex, a new, non-hierarchical architecture is taking place. Wright and Bourne [205] have called it a "heterarchical architecture." To some extent, hierarchical systems have the disadvantage that whenever two components, say a conveyor system and a machine tool, need to communicate (for instance to transfer a part from either side), the request must be sent through the lowest common controller. As shown in Figure 4.2, this is at the cell layer. Should the cell controller not acknowledge the request (e.g., due to network failure), the system would collapse.

In heterarchical control the message passing between layers is minimized since any controlling unit can communicate with another unit whenever it is necessary. This increases reliability. Also, each controlling unit is built with redundancy to assume the functions of other controllers. Thus, shop floor controllers can communicate directly with any, or resume control onto, other devices in the

system. The main disadvantages are increased overhead and software complexity. Object-oriented programming can reduce the latter (section 3.6).

Because PPC is inherently hierarchical, the heterarchical architecture does not change the way the functions are executed. It may, though, imply a different approach to software design. For instance, a knowledge-based controller would have to be able to access the knowledge of other controllers.

4.3 The Scheduling Problem

In section 3.4 we introduced the scheduling problem and examined some approaches by using AI-based methods. Yet, we have not discussed how the problem is originated, nor have we analyzed the strength and weakness of alternative methods based on Operations Research (OR) and Simulation.

We point out that OR, Simulation, and AI are not substitute tools but rather complementary methodologies. In fact, as seen in section 3.4 some knowledge-based systems have acquired their knowledge from mathematical models. The extent at which the three methodologies are used depends on the detail sought to model the production operations.

In this section we explore further this view and investigate the fundamentals of the approach used in this research, as introduced in section 3.6.

The term scheduling is used to refer to several produc-

tion planning functions, performed over different planning horizons, at the facility, shop, and cell layer, as shown in Figure 4.2.

We distinguish three scheduling levels. The first level develops a master production schedule; the second level determines the material requirements and allocates capacity; the third level deals with sequencing, dispatching, and routing on the shop floor. We refer to the last tasks as short-term scheduling.

We briefly discuss the first and second levels to explain how the third level gets the requirements for batch processing. In section 4.4 we describe the short-term scheduling.

4.3.1 Level 1: Master Production Schedule (MPS)

The MPS works at the facility layer and generates a production plan for the longest planning horizon (i.e., a year or a season). The MPS must meet the demand forecast for aggregated families of end-items (final products) and it is constrained by the production capacity during the horizon. The output is the production targets for families of products per period (e.g., per week).

4.3.2 Level 2: Material Requirements and Capacity Planning

This function generates disaggregated plans at the shop layer. The production target for a family of products breaks down into weekly requirements for individual end-items. The

output is the net requirement and timing for all manufactured and purchased components. The net requirement of a component is the actual production needed after balancing the gross requirement, inventory, and work in process. Its timing depends on the estimated manufacturing lead time.

The programs to calculate the net requirements can be run with a fixed frequency (e.g., every week) or with a variable frequency (whenever necessary). In either case the length of the planning horizon is the remaining number of periods since the current run.

Concurrently, capacity planning allocates shop capacity to the production schedule derived from material requirements planning. When this phase is carried out after the calculation of the requirements, the plan may be unfeasible and corrections must be made. The order size and timing generate the production targets for the next layer, the cell.

The most widely used method in these two levels is any computer implementation of the Material Requirements Planning (MRP) approach as originally described by Orlicky [138].

MRP is often criticized for its lack of a true planning capability since the detailed plan for individual items, as determined by the explosion through the bill of material, leaves the capacity analysis and the selection of lot sizes

to the scheduler. A feasible schedule is then found by "trial and error."

The so-called "regenerative MRP" is a rough form of feedback processing when some events affect an existing schedule. Ideally, this method would work in real-time. Changes in order quantity, modification of the bill of material, momentary reduction of capacity on the shop floor, and shortage of components, are the typical events that cause re-planning of net requirements.

This particular problem is a novel application for an expert system (see Steffen [176] for research in progress). The process of finding which items need re-scheduling (because not all the items may be affected) is not yet fully automated and the scheduler must use his knowledge to reduce the possibility of a full new run. However, the problem is formidable when an attempt to optimize the schedule is made.

First, it is known that MRP schedules are sensitive to a number of factors. For instance, (a) the lot sizing technique, (b) safety stocks, (c) formula to calculate the lead time, (d) randomness of the lead time, (e) scrap allowances, (f) regeneration frequency, (g) dispatching rules, and (h) structure of the bill of material.

Second, although several mathematical and simulation models have been constructed to analyze the effects of factors (a) through (g), there are not definite results

because of the disparity in MRP installations and the levels within factors.

Yet, existing studies can be good sources to acquire knowledge and build ESs. Berry [17], Blackburn and Millen [20,21], Chand [37], Kropp and Carlson [106], Moily and Matthews [127], Yano and Carlson [207] have investigated lot sizing models in MRP. Afentakis, Gavish, and Karmakar [4] modeled a MRP system and proposed a large-scale decomposition method. Thomas and Weiss [187] modeled the capacity requirement problem. Benton and Srivastava [14] investigated the effect of the bill of material structure. Marlin [118] designed an MRP simulator in SIMSCRIPT. Pentico [142] investigated the scrap problem. Russell and Taylor [156, 157] investigated dispatching rules.

The master production schedule, material requirements, and capacity analysis can be simultaneously modeled by using Hierarchical Production Planning (HPP) (Hax and Meal [79], Hax and Candea [80]).

In HPP, the process starts at the MPS level with a multi-item capacitated lot size model [19]. A mixed, integer, linear programming model subject to capacity, and demand constraints, determines the optimal production schedule for families of products during the planning horizon.

At the shop layer, families are broken down into individual items and a more detailed model is solved. In each decomposition, the production quantities from the higher

level become constraints in the next lower level. The scheduling problem is thus decomposed until it reaches the individual cell.

There is still a third method, Optimized Production Technology (OPT) [68]. OPT uses a network of queues model to analyze the effect of schedules, capacity, and bottlenecks on the shop floor. However, it is an "evaluative tool" (Suri [178]). It does not generate plans nor executes control but it analyzes their effects.

In recent years, the concept of Just-in-Time (JIT) manufacturing has attracted some attention. JIT manufacturing [128] is rather a philosophy to (a) minimize inventories, (b) reduce waste of material, labor and equipment, (c) maximize quality, and (d) allow personnel development [128]. As such, JIT is not a substitute for MRP (nor any of the other planning tools), but a complement. (Some production control methods such as KANBAN or "pull-systems" [115] are often associated with JIT. However, these are specific shop floor control techniques and we do not address them in the research.)

4.3.3 Level 3: Shop Floor Scheduling

This level schedules production at the factory floor. The requirements for end-items and components are grouped into batch jobs. Suppose there are M batches $\{B(1), \dots, B(i), \dots, B(M)\}$, each with $R(i)$ units of part type i . The total number

of parts in the mix is $\sum R(i)$ ($i=1,\dots,M$). Process planning has determined the machining and assembly operations and the machines where these operations can be performed. If each part type i has $O(i)$ different operations, the total number of activities to be scheduled is $\sum O(i) \times R(i)$ ($i=1,\dots,M$).

The shop floor scheduling problem. It decides when each activity starts and on what machine. Each part type may have its own precedence constraints. Alternative routings may exist. Machines and tools are subjected to failures. There exist transportation times between machines. There exists a setup time (e.g., tool change, NC program downloading) between parts of different type. Setup time can be dependent on the sequence. Each machine may have input and output buffers to store work in progress, with limited capacity (may be only to hold the part being worked). Pallets, fixtures, and tools are shared by all part types, and exist in limited quantities.

There exists several indicators to measure the performance of a schedule. The most common are tardiness, mean flow time, makespan, and equipment utilization. Figure 4.3 shows their definitions.

Considerable research in traditional manufacturing systems, in particular flow-shop and job-shop systems (described below), has produced results for only some

Notation

k	= job index
$A(k)$	= arrival time of job k
$T(k)$	= completion time of job k
$DD(k)$	= due date of job k
N	= number of jobs completed
CJ	= set of completed jobs
$L(k)$	= lateness of job k
	$T(k) - DD(k)$
NJT	= number of jobs tardy
	$\sum_{k \in CJ} \delta(k) , \quad \delta(k) = \begin{cases} 1 & \text{if } L(k) < 0 \\ 0 & \text{otherwise} \end{cases}$
r	= resource index
$U(r,t)$	= 1 if resource r being used at time t 0 otherwise
$OT(r)$	= time resource r remains operational

Formulas

$$\text{Mean flow time} = \frac{1}{N} \sum_{k \in CJ} [T(k) - A(k)]$$

$$\text{Mean tardiness} = \frac{1}{NJT} \sum_{k \in CJ} \max(0, L(k))$$

$$\text{Makespan} = \max_{k \in CJ} \{ T(k) \}$$

$$\text{Utilization } (r) = \frac{1}{OT(r)} \int_0^{OT(r)} U(r,t) dt$$

Figure 4.3 : Measures of Performance in Scheduling.

simplified cases of the problem above. In the remaining sections a general view of these approaches and their limitations is discussed.

4.4 Combinatorial Methods for Short Term Scheduling

Flow-shop scheduling. All part types visit the workstations in the same order and the routes are unique; all machines are free at the start and will remain operational during the planning horizon; machines do not require setup between part changeovers; transportation times are negligible; availability of material handling devices is disregarded; parts can accumulate in the cell; and all processing times are deterministic.

In this problem, closed-form solutions are only known for up to three parts and N machines (Johnson's Rule); the objective is to minimize the makespan (Figure 4.3).

Optimal solutions for any number of parts and machines can be found by combinatorial optimization using branch-and-bound (BAB) methods. The formulation by integer programming (see for instance, Bellman, Esogbue and Nabeshima [11]), suffers a rapid increase in the number of 0-1 variables. Thus, even this simple case may be difficult to solve.

Job-shop scheduling. It has the same assumptions and limitations of the flow-shop problem except that the scheduling is more complex because each part may have its own sequence. Conflicts in resource assignment occurs when seve-

ral activities compete for the same machine, robot, or material handling device. A decision must be made to decide what activity precedes the others when the conflict arises.

Florian et al. [62], developed efficient BAB algorithms to minimize the makespan. The authors, however, suggest the simulation of a subset of sequences when the setup time between activities cannot be neglected, setup time depends on the sequence, or measures other than makespan are used.

More recently, Shaw [166] developed a version of the job-shop scheduling for FMSs by using the A* algorithm (described in section 2.2.3). He designed a knowledge representation scheme to describe the precedence relations and to control the search.

Flow-shop and job-shop sequencing problems are known to be non-polynomial (NP) complete. Hence, the computing time increases exponentially with number of activities [11].

Although NP completeness explains the limitations of combinatorial optimization in large-scale problems, the real weakness arises from the assumptions of the models. For instance, the typical flow- and job-shop models assume that alternative routings do not exist and processing times are independent of the sequencing. In some FMSs, redundancy can be built by either grouping several machines with equivalent machining capability (pooling), or by setting up the machines with a magazine of different tools so they can

perform several operations. This allows alternative routings and increases reliability.

In real-time, the schedule is affected by the randomness of processing times, machine and tool breakdowns, and transportation delays.

4.5 Methods to Reduce Complexity

In real-time operations, each part must be traced individually. For batches containing many parts, the scheduling of each unit can make the problem computationally intractable.

Some simplification to reduce the number of activities to be scheduled can be achieved by sequencing only a sample. The sequence is then repeated over time. It is assumed that the machines keep the same tooling during the production of the mix. Two methods are described next.

Let,

- N = number of part types in the mix,
- M = number of machines of all types,
- $R(i)$ = requirement of parts type i ($i=1, \dots, N$),
- $P(i, j)$ = processing time of part i in machine j
($j=1, \dots, M$),
- $M(j)$ = number of pooled machines type j .

4.5.1 Periodic Scheduling

The sample is determined by using the idea of "minimal part sample." This is smallest possible set of parts in the same proportion of the total mix. For instance, if the mix

is $R(1)=2000$, $R(2)=1000$, $R(3)=3000$, then the minimal part sample is 2, 1, and 3 units of parts 1, 2, and 3 respectively. The sequencing is then reduced to six parts (approximation is required for non-integer values). Thus, parts are produced in a fixed proportion. This may be needed, for instance, to meet assembly requirements.

4.5.2 Production Ratios

The sequence of a sample must be determined in such a manner that the work load of the workstations is balanced (Stecke [175]). That is, the time allocated to produce the sample is the same for all workstations. If all workstations remain operational during the production of the batches, the balancing of work load implies equal utilization. The ratios are found by solving the linear system:

$$\sum_{i=1}^N P(i,j) \cdot X(i) = M(j) \cdot L \quad (j = 1, \dots, M), \quad (4.1)$$

with $X(i) > 0$.

Where, $X(i)$ is the production ratio of part i , and L is a dummy variable. For $N > M-1$, equation (4.1) may have infinite solutions. A generating set is obtained by fixing the value of $N-M+1$ variables. In general, the ratios will be fractional and rounding is necessary.

For instance, if after solving (4.1) we obtain $X(1) = 1$, $X(2) = 2$, $X(3) = 1$, then the sequences that balance the work

load can be: 1221, 2211, 1122, etc. That is, per each unit of part type 1, there are two units of part 2, and one of part type 3.

Equation (4.1) does not have a solution if the maximum processing time of all parts occurs on the same machine type. The machine group becomes the bottleneck and it is not possible to balance the work load.

The two methods above, however, are static. Real-time control may need to adjust the ratios when randomness in processing times, equipment failures, and transportation delays, cause the parts to fall behind schedule.

If a machine fails and there are part types which do not use the machine in their routing, those parts can still be dispatched while other parts are preempted. If there is a pool of machines and one machine fails, the affected parts can still be produced but at a lower rate. When the machine resume operations and capacity is restored, the control system must allocate more capacity for those parts behind schedule.

Stecke [175] suggested knowledge-based systems for this purpose but this has not yet been developed. In section 4.7 we will discuss another model to find adaptive ratios.

4.6 Dispatching Rules and Simulation

At any time t , there are in the shop several parts in different levels of completion. As the parts move through the shop, they need access to a resource (machine or han-

dling device) for the next operation. The parts are assumed to belong to a job, as shown in Figure 4.4. The boxes drawn in dashed lines represent the parts, the circles represent the operations, and the full tree (job) structure resembles the bill of material. There can be several jobs awaiting or in progress.

A dispatching rule determines the next part that access the resource according to a priority indicator. The indicator either uses information concerning the part only or information related to the job a part belongs. When the resource is busy, incoming parts assigned to it are held in a queue (logical or physical). Whenever the resource becomes idle, the part with the next highest priority is processed.

There exist static and dynamic rules. In the first type, the priority uses information that remains constant as the job progresses. In the second, the priority considers real-time information on the job's progress.

Figure 4.5 (a) and Figure 4.5 (b) describe common static and dynamic rules. Figure 4.6 shows the formulas for the indicators needed in the selection of the part with the highest priority. Figure 4.7 (a) shows the formulas to calculate due dates of the jobs and subassemblies. Figure 4.7 (b) shows the indicator used when sorting the parts and Figure 4.8 shows the effect and characteristic of each rule.

New priority indicators can be constructed by using linear combinations of those shown in Figure 4.7 (b).

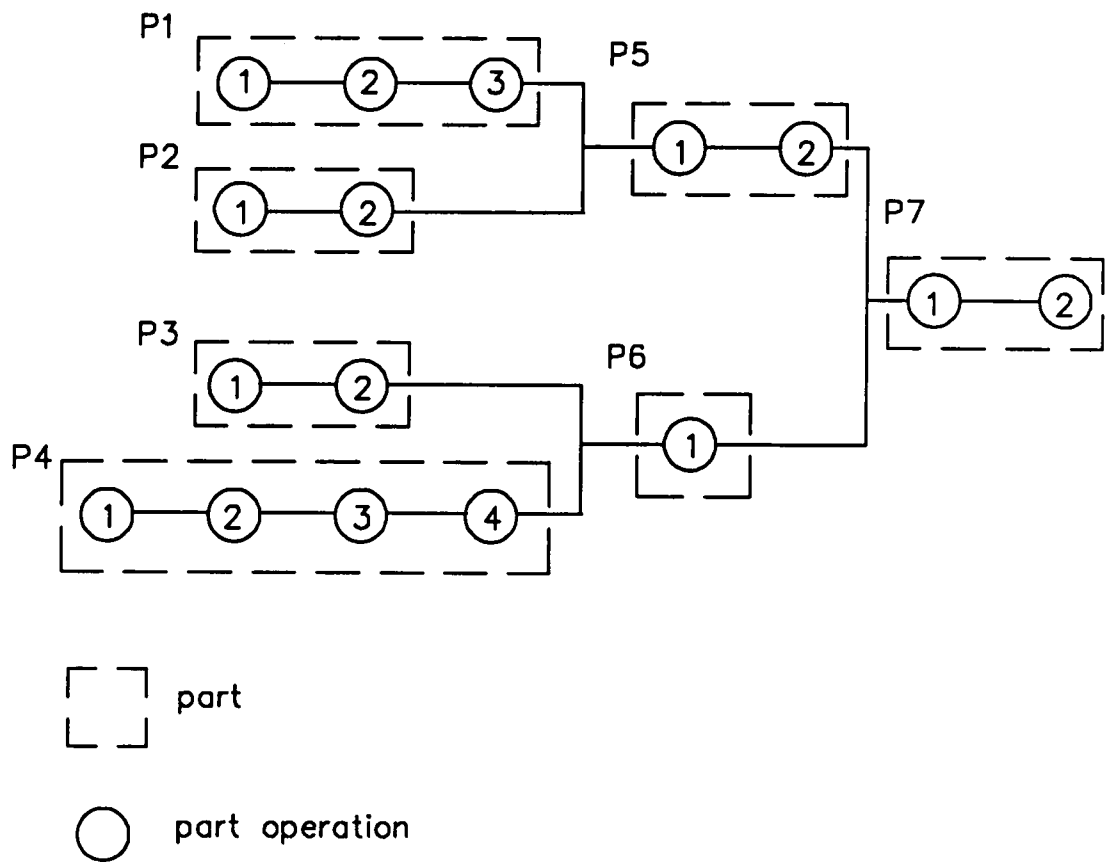


Figure 4.4: Example of Job Structure for Using Dispatching Rules.

STATIC RULES

Rule name	Action
First come first served (FCFS)	Parts are processed in the order the job arrives to the system.
Shortest processing time (SPT)	Selects the part with the shortest processing time among of all parts assigned to the resource.
Due date (DD)	Selects the part with the earliest assigned due date given by the job it belongs.

(a)

DYNAMIC RULES

Rule name	Action
Remaining total work (RTW)	Selects the part with the least remaining processing time among all parts.
Remaining number of operations (RNO)	Selects the part with the least remaining number of operations among all parts.
Shortest remaining path (SRP)	Selects the part with the shortest path towards the end-assembly
Branch slack with (BS)	Selects the part in the branch with least slack.

(b)

Figure 4.5: Description of Dispatching Rules.

NOTATION

k = job index ($k = 1, \dots, K$)
 $I(k)$ = number of parts in job k
 i = part index ($i = 1, \dots, I(k)$)
 $O(i)$ = number of operations of part i
 j = operation index ($j = 1, \dots, O(i)$)
 $A(k)$ = arrival time of job k
 $P(i, j)$ = processing time of j -th operation of part i
 $S(i)$ = set of operations of part i
 $PT(i)$ = total processing times of part i ,

$$PT(i) = \sum_{j \in S(i)} P(i, j)$$

$U(k)$ = set of uncompleted parts in job k
 $V(i)$ = set of uncompleted operations in part i
 $N(k)$ = number of operations in job k
 $C(k)$ = number of uncompleted operations in job k
 $Q(i)$ = remaining time for part i ,

$$Q(i) = \sum_{j \in V(i)} P(i, j) \text{ for } i \in U(k)$$

$J(i)$ = parent part of part i
 $RPA(i)$ = remaining processing time of part i until final assembly is completed

$$RPA(i) = Q(i) + PT(J(i)) + PT(J(J(i))) + \dots + PT(1)$$

$X(i)$ = set of components of part assembly i
 $BS(m)$ = branch slack of assembly part m

$$BS(m) = Q(m) - BDD(m)$$

$DD(k)$ = due date of job k
 $BDD(m)$ = due date of assembly branch m

AF = allowance factor to calculate job due date

Figure 4.6: Indicators Used in Dispatching Rules.

DUE DATES

$DD(k)$ = due date of job k , based on work content

$$DD(k) = A(k) + AF \cdot \sum_{i \in C(k)} PT(i)$$

$BDD(m)$ = due date of assembly branch m , based on work content

$$BDD(m) = \begin{cases} A(k) + AF \cdot \sum_{s \in X(J(s))} PT(s) & \text{if } X(m) = \phi \\ \sum_{i \in X(m)} BDD(i) + AF \cdot \sum_{s \in X(J(s))} PT(s) & \text{if } X(m) \neq \phi \end{cases}$$

(a)

SORTING CRITERIA

Rule	Indicator *
FCFS	$A(k)$
SPT	$PT(i)$
DD	$DD(k)$
RTW	$Q(i)$
RNO	$V(i)$
SRP	$RPA(i)$
BS	$BS(m)$

* Ascending order.

(b)

Figure 4.7: Due Date Calculation and Sorting Criteria.

RULE	CHARACTERISTIC
FCFS	Requires minimum information (used as reference).
SPT	Minimizes mean flow time in single machine.
DD	Minimize mean tardiness in single machine.
RTW	Prioritizes those parts near completion, based on remaining time.
RNO	Same as above but based on number of operations remaining.
SRP	Prioritizes parts in sub-assemblies closer to end-assembly.
BS	Tends to minimize mean lateness of subassemblies.

Figure 4.8: Characteristics of Dispatching Rules.

For instance, SRP with BS coordinate the completion time of subassemblies to reduce the delay in the start of the parent component. RTW with RNO speed up parts near completion, based on time or number of operations remaining.

Job-shop dispatchers have used these rules for decades in semi-automated mode. Programs determine the priorities and the dispatcher reads the computer printouts or terminals on the shop floor. In CIM, both tasks can be fully automated.

Shaw [167] developed an implementation of dispatching rules in a cellular manufacturing system (section 4.2) with identical processors (cells) linked by LAN. Although the rule applies to the selection of the cell instead of the part, this case still falls into the dispatch category.

When a processor completes a task, it announces a request so another processor performs the remaining operations. A computerized cell manager, using the LAN, calls for "bids" from all the cells that can perform the task (including the one releasing the task). Each bidding cell offers the earliest completion time when the new task can be finished, considering its current assignments. The cell manager then selects the lowest bid and either sends the part to the cell or puts it in a logical queue.

For traditional job-shops there exist many studies attempting to find general relationships to decide which dispatching rule performs best in one effectiveness measure, namely, due date, resource utilization, tardiness, or mean

waiting time (see Russell and Taylor [156,157] for most recent investigation).

Among the factors affecting the rule selection, the end-item structure (as given by the bill of material or assembly net), the method to calculate the due date, and the criteria to assign resources, may all be significant. These findings are derived from experimental design on simulated samples.

In the single-machine problem, the mean flow time (as defined in Figure 4.3) is minimized by ranking tasks in increasing order of their processing time (the SPT rule). Mean tardiness is minimized by ranking tasks in increasing order of their due dates (the DD rule) (Rachamadugu [147]). As a starting point, this suggests that SPT is a good heuristic to minimize mean flow time and to maximize production. Whereas, DD is a good choice to complete jobs by due date. However, none of both guarantees that the two objectives are met. In any case, these hypotheses need to be tested by simulation experiments.

General relationships, or definite results, are difficult to obtain because of the variety of facility layouts, products, and resources. However, for a specific facility this experimental methodology should lead to useful results.

The main limitation of dispatching rules is that long term performance in complex systems involving a network of

resources must be analyzed by experimental design on simulation runs.

Although, dynamic dispatching rules use real-time information about the job progress, they are static during run-time. Changing the dispatching rules (i.e., adapting them to the real-time status of the facility) has not been investigated. An expert system seems to be appropriate for this purpose. It may be even possible to use simulation (to quantify the heuristic function) and predict the effect of changing the rule in real-time. But, the computation could be slow. Wysk et al. [206] discuss this possibility but there are not enough results to conclude if this is effective. Russell and Taylor are investigating the use of an expert system to encode the results of their experimental designs. However, the testing will require a tool as the one developed in this research.

4.7 Control Theory

Given the mix of parts to be produced over a planning horizon of length T , optimal control models find the optimal trajectory of the production in function of time.

Recent developments by Akella, Choong, and Gershwin [5], Gershwin et al. [71,72,73], Kinemia and Gershwin [98], have produced interesting results. Let us define the following notation:

N : number of different parts in the parts family (mix),

$R(j)$: requirement of part type j ($j = 1, 2, \dots, N$),

$[0, T]$: period of time to produce all the mix,

$X(j, t)$: buffer level of finished parts type j , at any time t , $t \in [0, T]$,

$U(j, t)$: cumulative production of part type j in $[0, t]$,

$D(j, t)$: cumulative demand of part type j in $[0, t]$,

$X(t) = U(t) - D(t)$, with X, D, U being vectors,

$u(j, t)$: production rate of part type j ,

$$u(j, t) = \frac{d U(j, t)}{dt}$$

$d(j, t)$: demand rate of part type j ,

$$d(j, t) = \frac{d D(j, t)}{dt}$$

$$D(j, t) = t \cdot d(j, t) \quad \text{if } d(j, t) \text{ constant,}$$

$u(t)$: vector of $u(j, t)$ components ($j = 1, \dots, N$),

$d(t)$: vector of $d(j, t)$ components ($j = 1, \dots, N$),

$A(t)$: vector of machine states,

$$A(t) = [a(1, t), \dots, a(M, t)],$$

$$a(i, t) = \begin{cases} 1 & \text{if machine } i \text{ is operational at } t, \\ 0 & \text{otherwise} \quad (i = 1, \dots, M) \end{cases}$$

$R[A(t)]$: convex region where $u(j, t)$ is feasible,

$g[X(t), A(t)]$:
functional that accounts for the cost of
having the buffer $X(t)$ given the machine
state $A(t)$.

The optimal control model is then given by:

$$\text{Min} \quad \int_0^T g[X(t), A(t)] dt \quad (4.2)$$

$$\text{subject to} \quad \frac{d X(t)}{dt} = u(t) - d(t) \quad (4.3)$$

$$u(t) \in R[A(t)] \quad (4.4)$$

with $u(t)$, $d(t)$ continuous; $X(t)$ differentiable.

In principle, the rate at which the parts are dispatched into the cell would follow the demand rate. However, random machine failures cause temporary decreases in capacity and such a policy may lead to starving processes somewhere downstream. Therefore, some surplus production must be held in the buffer $X(t)$ to prevent against temporary decreases in capacity.

Thus, the capacity of the system is variable and depends on the number of operational machines that can process the parts. The planning horizon is then divided into shorter intervals (of random length) where the machine states and the production rate remain constant. When the state of the machines changes (from operative to non-operative, and vice versa) the production rates must be recalculated to take into account the new capacity available.

Kinemia and Gershwin [98] proved that an approximation to the objective function (4.2) subject to constraints (4.3) and (4.4) is given by the LP:

$$\text{Min } \sum_j \frac{\partial g_j}{\partial X_j} u(j,t) \quad (4.5)$$

$$\text{subject to } \sum_j P(i,j) u(j,t) \leq M(i,t) \quad (i=1,\dots,M) \quad (4.6)$$

$$u \geq 0$$

Where $M(i,t)$ is the number of operational machines type i at time step t ; $P(i,j)$ is the processing time of part j in machine type i .

The objective function (4.5) is an approximation to (4.2) obtained by expanding (4.2) in a series, dropping second order terms, and using equality (4.3). The convex set of linear constraints (4.6) is the set $R[A(t)]$ in (4.4).

If $g(\cdot)$ is assumed to be a convex quadratic form, the partial derivative in (4.5) becomes a linear cost coefficient $C[j, X(j,t)]$. This is constant given a time t but changes with $X(j,t)$. $C(\cdot)$ penalizes the excess or deficit in the buffer state $X(j,t)$ with respect to a threshold value $H[j, A(t)]$. This value is called the "hedging point" and it is the optimal buffer level to prevent for sudden losses of capacity. It depends on the machines state $A(t)$. By solving (4.5) the production rates and buffer level, by using

equality (4.3), can be determined. The lower level of dispatching and routing parts within the cell must assure the trajectory $X(t)$ is met.

4.8 Summary

In this chapter we have seen the major functions that are executed in a CIM system. The hierarchical schema has allowed us to visualize the purpose and scope of each of the production planning and control functions in an organized manner. The idea of hierarchical planning let us breakdown a large problem onto smaller problems for which specialized methods can be applied. A hierarchical decomposition is not only bottom-down. Information feedback must be generated in order to adjust the plans provided for the higher level when the lower level cannot meet the goal.

The hierarchical schema is also useful to integrate the PPC functions with the computer control architecture of the CIM system. For specific equipment control, a combination of hierarchical and heterarchical control may be necessary, as discussed in section 4.2.3.

For real-time control at the shop floor level, our Cell Manager (CM) could be implemented by two approaches. One is to obtain long-term results by using simulation and then apply the dispatching rules to manage the actual operations in the cell. The other is to use the reduction method by using production ratios, as seen in section 4.5.2. Since the

ratios are static and do not consider random capacity losses, a new method that combines ratios with the hierarchical control model proposed by Gershwin et al. must be tested.

The simulation approach does not tell us how to allocate or re-allocate capacity whenever drops in capacity occur nor guarantees that the dispatching policy is feasible. If real-time capacity is ignored, as dispatching rules do, parts can be introduced at a rate such that queues (inventory in process) could develop. Tuning of simulation parameters would be required to determine what dispatching policy keeps low work in process and within the cell capacity. We would have to apply heuristic rules to decide when to load a part in the cell based on the simulation results or based on standard practices. For instance, the Kearney & Trecker rule is (adapted in rule format, see [96], pp.7.11):

```
"if (a production requirement exists) and
    (a free cart is available) and
    (a fixture is available) and
    (a material is available)
then
    (introduce a route for the part)."
```

A dispatching policy based on the control model would let us re-distribute capacity among parts and feasibility is guaranteed. However, the control model only provides information concerning the dispatching policy. Yet, various problems remain:

1. If transportation delays are ignored the production rates can be overstated and parts would be introduced at a rate which may be infeasible. This can be fixed by subtracting from the right-hand side of the constraints (4.6) an estimate of the time lost in moving parts.
2. In the manufacturing cell considered in this research, the operation times and output rates may not be as small as to assume a good match between the continuous, smooth trajectory given by the model and the ladder-shaped actual trajectory. This means that a simple, periodical comparison of the actual and theoretical trajectory is not always possible. A match-up procedure then would show its results with some time lag.
3. Time spent in moving parts between stations affects utilization and due dates. The traffic must be managed in such a way that transportation delays are minimized.
4. Machine failures must be handled in detail. Although we do not address the diagnosis problem (which can be implemented by using the CM shell), failures may occur while processing parts, or when parts are on the way to the station. When resuming operations, those parts have be recovered (some may be discarded).

The issues above indicate that both the control model and the AI-based approach must cooperate in order to handle a number of situations that arise during actual operations. In

Chapter 5 we discuss the architecture of CM and build the knowledge base required to handle these problems. In Chapter 6 we conduct experiments to test the CM's effectiveness.

CHAPTER 5

RESEARCH METHODOLOGY

A control knowledge-based system must continuously monitor state changes in the system under control to prescribe and execute proper actions. To meet this goal an appropriate architecture (components and interactions) is designed for the FMS described in Chapter 1.

In this chapter we focus on the architecture of CM. We discuss the knowledge and data representation, inference engine, the knowledge base, and supporting algorithms. The interaction between these components is described prior to conducting experiments in Chapter 6.

5.1 Event-driven Inference

Suppose we denote the structure of the control expert system by $E = E(X, K, O)$. Where, X is the system being controlled, K is a knowledge base in X , and O is the set of objects in X . The set O consists of objects $\{O(1), \dots, O(i), \dots, O(p)\}$. Each $O(i)$ can be described by a n_i -dimensional array of discrete or continuous attributes $A(i) = [a(i,1), a(i,2), \dots, a(i,n_i)]$, with $A = [A(1), \dots, A(i), \dots, A(p)]$ being the status of all objects. Without loss of generality, each $a(i,j)$ is a function of time $f_{ij}(t)$. Thus, the status of the object $O(i)$ is a function of the time t , denoted $S(A(i,t))$.

The status of the system is denoted $S(A(t))$.

The status S_x of the system X at a time t_k , as given by $S(A(t_k))$ is:

$$S_x(t_k) = S(A(t_k)) = \phi (S(A(t_{k-1}), E)) \quad (5.1)$$

$k = 1, 2, \dots, K$; $S_x(t_0)$ given.

Equation (5.1) states that the system X changes its status at discrete time steps. The function ϕ accounts for a state transformation between discrete time steps. ϕ depends on the last status of X and the structure of the control system. The transition from one state to the next is driven by events. The value of the attributes after a transition occurs is determined by the contents of the knowledge base K in the system X .

In order to perform a transition, we define event-driven inference as follows:

Event-driven inference. The inference engine of a control expert system $E = E(X, K, O)$ is said to be "event-driven" when there exists a finite set of events $\{ V_1, \dots, V_i, \dots, V_q \}$, known to affect the system status, and the state transitions are conveyed by a knowledge representation (KR) method.

In event-driven inference, an event management subsystem picks up the next event, interrupts the control system, and drives the inference through the knowledge base to deduce

and execute the proper actions. The primary criterion to process the events is the time of occurrence (a secondary criterion could be an event-dependent priority).

5.2 Architecture

CM has six components, shown in Figure 5.1: (a) interface, (b) database, (c) working memory, (d) knowledge base, (e) inference engine, and a (f) library of procedures. These are defined below:

Interface. The interface is a set of facility-dependent routines to pass messages between the shop floor and the CM in either direction. The implementation of these programs depends on the protocol, topology, and the manufacturing message formatting system of the LAN. In this research the interface is a conceptual construct and its details are left to a specific implementation. Two important conditions the interface must satisfy are that:

1. the real-time status of the cell is accurate, and
2. the actions derived from the CM can be mapped onto instructions to the programmable controllers.

The first condition is necessary to guarantee that the actual status (properties) of the objects is known at any time and it is error-free. The second condition can be achieved by decomposing a high-level command language into lower-level directives which can be transparent to the final

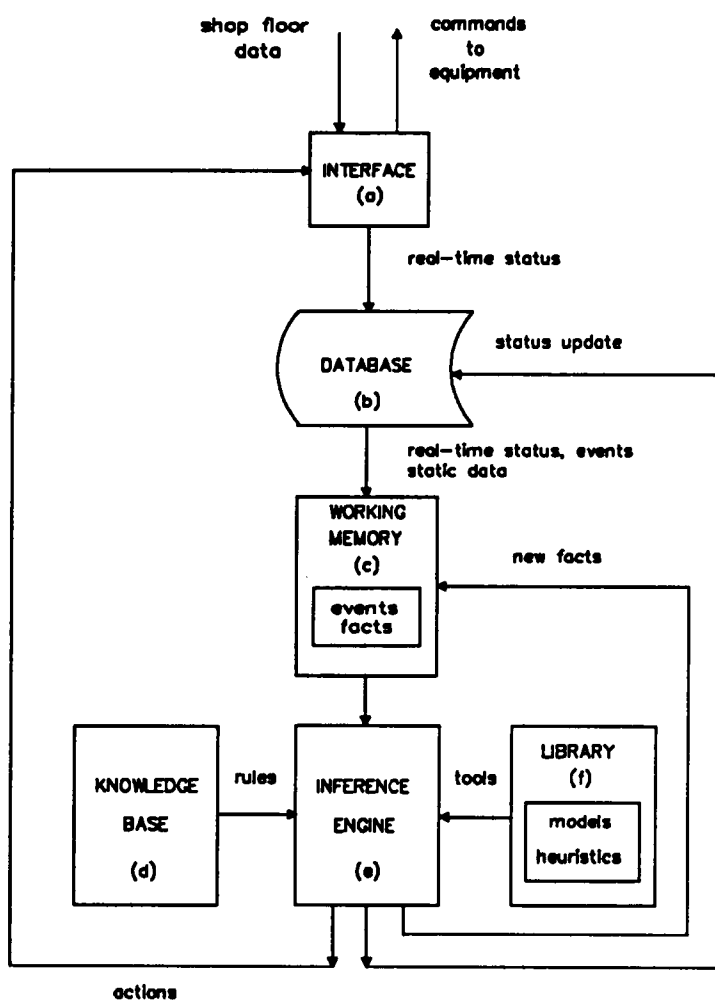


Figure 5.1: Architecture of CM.

application. An example of a message passing system onto which a high-level command language can be implemented, is the set of C-language functions developed for the Manufacturing Messaging Standard of the Manufacturing Automation Protocol (MAP) [89].

In the research we obviate the details of building the interface by generating simulated events and messages. The state transitions conveyed by the event-driven inference (as given by equation 5.1) generate, in turn, new simulated events and messages.

Database. The database stores a representation of the objects in the cell and can be updated by the CM inference engine, through the interface, or both.

Working Memory. The working memory is temporary storage allocated to place events and facts used by the inference engine while making deductions. Events are messages requiring the attention of the CM to take some action. Facts are assertions about the status of the objects in the cell or any message (phrase) that can be built based on the status of the objects.

Knowledge base. The knowledge representation (KR) method uses production rules, as introduced in section 2.4.2. The knowledge base contains rules to deduce new facts and per-

form actions according to the events that occur and the known facts when the inference is made.

Inference engine. The engine is an implementation of forward-chaining, pattern-directed search. Forward-chaining is adopted as problem solving model because the transition to new goal states is driven by events and known facts at the time the inference is made. Due to the large number of goal states that are possible after a transition occurs, exhaustive backward-chaining search is not suitable.

Library. The library is a set of algorithms or procedures that can be invoked by the inference engine to execute a process which would be computationally inefficient if done by symbolic processing. Examples of these processes are the A* algorithm described in section 2.2 with applications to vehicle routing, and a linear programming module to solve the model described in section 4.7. In general, the library is intended to accommodate any procedural task.

In order to describe the operation of the CM, we first summarize the syntaxis and usage of the knowledge representation scheme and then discuss in detail how this is applied to the operations in the manufacturing cell under study.

Some ideas about inference methods and knowledge representation are built on Chapters 18 and 22 of the book by Winston and Horn [204]. In the research some major modifications have been made in order to:

1. Create an event management mechanism that allows the implementation of the state transitions given by equation (5.1) above.
2. Include procedural attachments to rules.
3. Facilitate the binding of variables in the pattern-matching mechanism.
4. Allow rule-based inference from frame-based KR.

5.3 Object Representation

Objects are represented in frame-like format, as introduced in section 2.4.5. To each object we associate a number of markers (relations, properties, etc.) to encode different pieces of information we may be interested in.

The inference in frame-based systems is to determine whether a particular object matches the characteristics of a predefined prototype. In our case, we use frames only to represent the objects since the inference is rule-based.

Still these objects may call for specific procedures to determine the value of some attributes but these procedures are invoked by the rules. As shown in Figure 5.2, the data organization for an object has a tree structure. The items in each node of the tree are defined below.

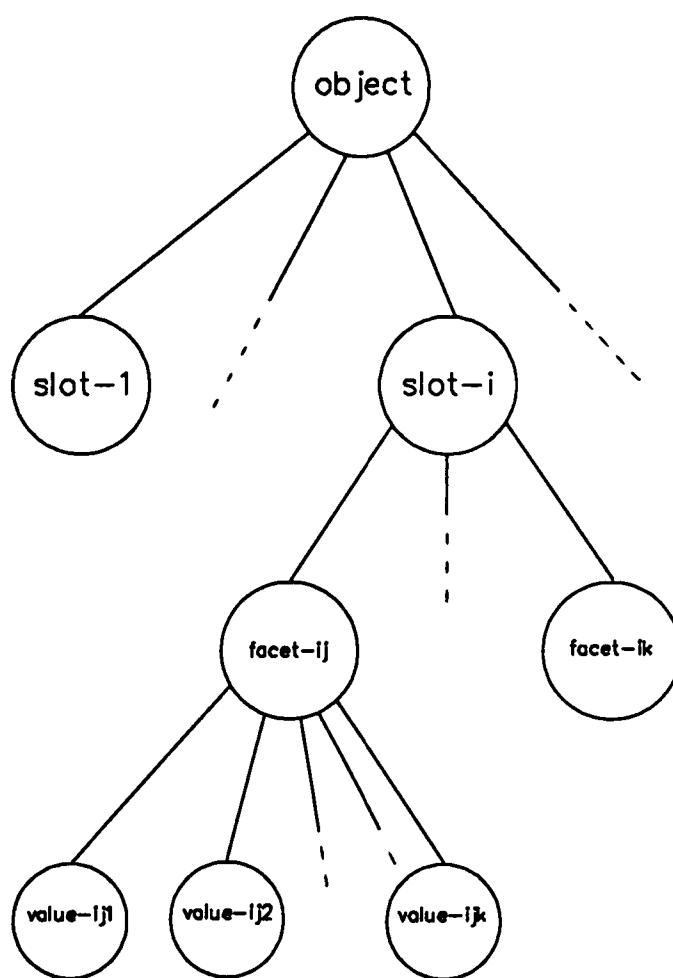


Figure 5.2: Graphic View of Object Representation

Terms associated with a frame. The following terminology is used:

1. Object: generic name of an object or class of objects which we want to describe.
2. Slot: any alphanumeric symbol to describe an attribute, or a relation associated with the object.
3. Facet: any alphanumeric symbol indicating an aspect of the slot.
4. Value: any data element associated with a facet of a slot--either alphanumeric symbol or list.

The tree shown in Figure 5.2 is represented by the structure below:

```
(defobject object
  (slot-1.....)
  .
  .
  (slot-i (facet-ij value-ij1 value-ij2 value-ijk)
    (facet-ik.....)))
```

(5.2)

Example 5.1 shows one form of representing the transporters of Figure 1.1, by using (5.2):

Example 5.1. Carts are dynamic objects whose status must be known at any time. We associate properties such as the cart's code, speed, status, position, last position, current task being executed, destination, current load, and a path when a cart is moving. The cart's frame at some time step is represented as follows:

```
(defobject cart
  (codes      (are (c1 c2 c3 c4)))
  (speed      (c1 100) (c2 100) (c3 100) (c4 100)
               (in ft-per-min))
  (status      (c1 stopped-idle)
               (c2 stopped-assigned)
               (c3 moving-assigned)
               (c4 moving-idle))
  (position    (c1 15) (c2 13) (c3 5) (c4 7))
  (last-position (c1 15) (c2 12) (c3 4) (c4 6))
  (task        (c1 stage) (c2 unload-machine)
               (c3 load-machine) (c4 moving))
  (destination (c1) (c2 3) (c3 6) (c4 8))
  (load        (c1 _) (c2 _) (c3 "P02-6") (c4 _))
  (path        (c1) (c2 (14 2 3)) (c3 (6))
               (c4 (8))))
```

Deriving facts from frames. The purpose of the object representation is to allow the inference engine to know what facts are true at any given time. The frame defined by form (5.2) above allows us to define a simple grammar given by:

Fact := (<object> <slot-i> <facet-ij> <value-ijk>) (5.3)

For instance, in the cart frame shown in Example 5.1 the fact (cart status c1 stopped-idle) is true.

Given an event at a time step, the mapping of frames onto facts and the selection of the necessary rules to perform the inference is driven by meta-knowledge. This meta-knowledge filters and brings into the working memory only the objects and rules associated with an event.

5.4 Rule Representation

The form (5.3) above plus any predicate-like structure in the working memory are the basic elements needed to write a rule.

The format of a rule is as follows:

```
(defrule <rule-name>
  if
    <antecedent-1>....<antecedent-n>
  then
    <consequent-1>....<consequent-m>)      (5.4)
```

Rule (5.4) means that if all antecedents are facts, then all the consequents are either facts or actions to be executed. Antecedents and consequents are made of arguments or tokens to express some condition or action in declarative form. Each token may be a constant or a variable. When using variables, rules serve as predicates to express universal formulas.

In AI terminology, a variable in a predicate is said to be "instantiated" when there exists a fact with identical structure to that of the predicate from which the variable

can take its value. All variables in a rule need first be instantiated in order for the rule to fire.

In a rule we also distinguish "directives" to perform a procedural task (e.g, a database update or running a numerical process). Example 5.2 illustrates a rule which includes several features widely used in the knowledge base for the experiments of Chapter 6. This knowledge base is fully listed in Appendix C.

Example 5.2. When carts finish performing a task, a message is passed to the CM indicating that the cart is free at some location (logical unit). New tasks have been scheduled earlier as requests for transportation at the workstations. These requests are served on a first-come-first-served basis by the nearest cart available. Thus, when the cart finishes a task, it can be moved to another location to attend an earlier request on hold (because all the carts were busy). Or, if no request exists, the cart can be sent to a staging area to wait for new assignments, or simply wait at the current position idle.

Requests arise at any time and those that cannot be attended immediately are placed in a logical queue until a free cart can pick them up. The rules to handle this situation are as follows:

```
;;; rules when cart is free
```

```
(defrule free-cart1 (5.5)
```

```
if
```

```
  (free-cart ?tnow ?cart at ?pos)
  (queue cart-assignment ?cart ?queue
    (has-value ?queue))
  (<- (?task ?dest ?load)
    (release-from-queue cart-assignment ?cart))
  (<- ?path (get-path ?pos ?dest))
```

```
then
```

```
  (-> (cart status ?cart moving-assigned
    task ?cart ?task
    destination ?cart ?dest
    path ?cart (rest-of ?path)
    load ?cart ?load))
  ($ (move-cart ?tnow ?cart from ?pos to ?dest))
  (exit))
;;;
```

```
(defrule free-cart2 (5.6)
```

```
if
```

```
  (free-cart ?tnow ?cart at ?pos)
  (queue cart-assignment any ?queue
    (has-value ?queue))
  (<- (?task ?dest ?load)
    (release-from-queue cart-assignment any))
  (<- ?path (get-path ?pos ?dest))
```

```
then
```

```
  (-> (cart status ?cart moving-assigned
    task ?cart ?task
    destination ?cart ?dest
    path ?cart (rest-of ?path)
    load ?cart ?load))
  ($ (move-cart ?tnow ?cart from ?pos to ?dest))
  (exit))
;;;
```

```

(defrule free-cart3                                     (5.7)
  if
    (free-cart ?tnow ?cart at ?pos)
    (lu unloading is ?dock (~ = ?dock ?pos))
    (cart last-position ?cart ?lastp)
    (zone-id ?lastp ?pos ?zone)
  then

    (if
      (queue zone-queue ?zone ?queue
        (has-value ?queue))
      then
        (-> (cart status ?cart stopped-idle))
        ($ (release-cart ?tnow at ?pos)))

    (if
      (queue zone-queue ?zone ?queue
        (has-no-value ?queue))
      then
        (-> (cart status ?cart stopped-idle
          task ?cart
          load ?cart _))))

  ;;

(defrule free-cart4                                     (5.8)
  if
    (free-cart ?tnow ?cart at ?dock)
    (lu unloading is ?dock)
    (lu staging is ?stage-pos)
  then
    (-> (cart task ?cart stage
      status ?cart moving-idle))
    ($ (move-cart ?tnow ?cart from ?dock to
      ?stage-pos)))

  ;;

```

In rule (5.5) a cart becomes free at a location and the cart has a queued request. The entity released from the logical queue is matched with three variables to find the task, destination, and load. The path from the current position to the selected destination is determined next. Then, the cart parameters are updated with the retrieved values and the cart is scheduled to move to the next destination.

The (exit) predicate is a meta-logical directive to indicate that no more rules need to be searched if this rule has fired.

In rule (5.6) the request can be served by any cart that becomes free. The request could have been made, for instance, when no cart was available to attend it and was queued so any free cart could pick it up later.

In rule (5.7) if the cart is at a position different from the unloading dock, then the cart is stopped and remains idle at that position (e.g., after loading a machine). But, if the cart is blocking the zone then it is released.

In rule (5.8) the cart is freed at the unloading dock then it is sent to the staging area where it will remain stopped and idle. Whenever a new request arises, this cart may be assigned provided that no other cart is closer to the requesting location.

5.4.1 Variables and Directives

In Example 5.2, the rules were built with several variables and directives. An overview of these elements is presented next. A more comprehensive description of the CM's KR techniques and usage is provided in Appendix D.

Variable binding. Variables are specified by the prefix "?" For instance, ?tnow, ?cart, and ?pos are the variables in the antecedent (free-cart ?tnow ?cart at ?pos) of rule

(5.5). Where, ?tnow and ?pos are short names for "time now" and "position", respectively. There are three ways to instantiate a variable. First, by pattern matching with facts derived from frames, as in form (5.3). Second, by pattern matching with any fact asserted in the working memory. Third, with the directive "<-" which assigns the variable (or a list of variables) the evaluation of an expression. For example, the antecedent:

```
(<- ?path (get-path ?pos ?dest))
```

places in the variable ?path the value of the expression (get-path ?pos ?dest). This is a library function that searches for the path that minimizes the travel time between two points. The path is found by using the A* algorithm and it is determined based on the current status of the paths.

Filters. The function of filters is to perform logical tests on the value of instantiated variables. We can regard them as constraints for the variable. The filter (has-value ?queue), in rule (5.5) is an example of this. Any form can be evaluated provided that it returns either a true or a false flag. Filters are particularly useful in non-trivial tests and they have been incorporated in expert system shells (e.g., in ART [41]), though for a small set of logical operators.)

Frame update. Updates are performed with the directive " $\rightarrow$ " followed by the frame-slot-facet-value combination. For instance, an update to the cart frame in rule (5.6) is represented as follows:

```
( $\rightarrow$  (cart status ?cart moving-assigned
      task ?cart ?task
      destination ?cart ?dest
      path ?cart (rest-of ?path)
      load ?cart ?load))
```

Update functions can be used to remove slots and even full frames, if not longer needed. For example, while a part family is being produced a trace of each part code is necessary but the specific part is not longer needed once unloaded from the cell.

Event scheduling. A predicate preceded by the sign "\$" is meant to schedule subsequent events. In an actual setting this would be a command to be issued to the system. In the research, the event is scheduled in ascending order of its time of occurrence in the *events* list. Events can be preempted at any time by using the directive "preempt-event." Appendix D, section D.4, explains this.

General procedural calls. Two types of general procedural calls can be incorporated in a rule. First, functions can be called by reference to perform some action on the right-hand side of a rule. Second, functions can be called by value to

evaluate an expression within an antecedent or consequent. A function is treated as a predicate and it is simply called by a LISP expression of the form:

```
(function-name arg1 arg2 arg3....argn),      (5.9)
```

with `arg1`, `arg2`, ..., `argn` being arguments. Arguments can be constants or instantiated variables. In rule (5.5), the expression `(get-path ?pos ?dest)` is called by value to bind the variable `?path`. Examples of calls by reference are the predicates "collect" and "put-in-queue" in the following rules:

```
(defrule free-station2                                (5.10)
  if
    (free-station ?tnow at ?pos)
    (queue buffer ?pos ?queue (has-no-value ?queue))
  then
    (-> (station status ?pos idle))
    (collect station-util ?pos ?tnow 0))
  ;;;
```

```
(defrule cart-request3                                (5.11)
  if
    (cart-request ?tnow at ?pos for ?task ?load)
    (selected cart is ?object (~= ?object none))
    (<- ?cart (first ?object))
  then
    (put-in-queue cart-assignment
      (entity-is ?task ?pos ?load) ?cart))
  ;;;
```

Since some implementations of Common LISP support calls to foreign code, the functions could be written in any language if required (e.g., the VAX Common LISP support this feature with object modules generated by various languages such as FORTRAN, PASCAL, C).

5.4.2 Top-level Algorithm for the CM Inference Engine

Whenever a set of events $V = \{V_1, V_2, \dots, V_m\}$ occurring at a time t require the CM's attention, the inference engine searches the knowledge base according to the following algorithm:

Let,

R : set of all rules in K of $E(X, K, O)$, as defined in section 5.1,

O : set of all objects in $E(X, K, O)$,

t : time at which actions are requested (time step),

V : set of events at time t , $V = \{V_1, V_2, \dots, V_m\}$

$rules(V_i)$: set of rules associated with event V_i ,

$objects(V_i)$: set of objects associated with event V_i ,

$facts(objects(V_i))$: set of facts associated with $objects(V_i)$.

Algorithm.

Step 1. Get the set of events V , at time t .

Step 2. Do while $t < \text{run-time}$ and $V \neq \emptyset$: (5.12)

Step 3. For all V_i in V do:

Set rules and facts by,

rules $\leftarrow$ rules(V_i),

facts $\leftarrow$ facts(objects(V_i))

Step 3(a). Perform a forward-chaining inference.

This step will assert all new facts and execute all actions driven by event V_i .

Step 3(b). Next i .

Step 4. End do.

Step 5. Stop

In step 3(a), rules are traversed one at a time and all instances that match a rule are fired. Due to the construction of the engine, the firing may result in new facts, execution of a function, or updates to the frame database. These are indicated as outgoing flows from block (e) in Figure 5.1.

There exist two ways for traversing the rules. First, whenever a rule is fired, all rules can be reset to handle the case when the new asserted facts allow another rule, already traversed, to deduce a conclusion that was missed before because those facts did not exist. Second, the rules can be traversed by using a single pass if the previous situation does not occur.

5.4.3 Computer Operating Systems and CM

According to algorithm (5.12), events are processed in groups sorted by time (and a secondary priority criteria, if necessary). An operating system for real-time control must be able to manage new requests that occur while the processor is executing other tasks. An example of this case arises when the user (operator) requests explanations about facts and rules, or must respond to questions asked by the computer (information that cannot be obtained from the database). For instance, if a trouble develops in a machine, the operator may want to run a diagnosis expert system while CM continues processing other incoming events.

Conventional knowledge-based systems are static and multiple, parallel events, do not occur. Processing is done sequentially. Some expert system environments (e.g., ART [41]) allow so-called "utterance facts" to poll inputs from an external device (e.g., a mouse click in a user interface), which are processed by a procedure after all rules have been traversed. Although this can be programmed to accept other inputs (e.g., from a serial interface), the processing is still sequential, and not suitable for parallel events.

Young [209], distinguished two general strategies for software design in control systems. The first is the sequentially driven program in which requests are queued as they arrive and the second is an interrupt-driven, multi-tasking

system. The former has the disadvantage that does not provide for preferred service (in our case, the event priority). Thus, it would be inappropriate for our purpose. The second strategy does allow multi-tasking by letting an external device interrupt the task of the processor, save the state of the program being run, execute the other request, and then re-take the suspended task.

In an actual implementation, the CM requires a multi-tasking operating system. However, it is desirable that the handling of parallel events is transparent to the development of the knowledge base.

Implementation details concerning the multi-tasking nature of the CM is a subject for further applied research. In our experiments, however, interactions with the user can be incorporated without difficulty.

5.5 Description of the Simulated Cell

With the architecture and inference approach described in the preceding sections, the operation of the cell can be simulated in a closed-loop mode. That is, the CM receives messages about events, it performs the actions, and the actions generate new events. We now describe the operation of the cell.

5.5.1 Dispatching and Routing

A family of N different part types is produced in the cell shown in Figure 1.1. For each part type, the requirement is $R(i)$ ($i=1,\dots,N$).

Whenever the CM generates a load request, the parts are loaded onto carts at the loading dock (logical unit 1). The selection of the part depends on whether all the machines in the part type routing are in their operative status and also on the production level achieved at the time the part type is dispatched.

The actual dispatch of the part occurs after the first station in the route becomes available and a cart picks up the load and moves it to the station. If the CM receives a request for loading a part but the first station is assigned or non-operative, then the part is placed in a buffer at the station. How often the queuing will occur is a function of the dispatching rules we want to investigate. Parts are transported from one station to the next when the process ends.

If there is only one station for the next operation, the choice is obvious. If stations are "pooled" (i.e., a number of stations have the same capability to perform an operation), then a policy decision is required. For instance, Shaw [167] investigated the shortest processing time (SPT) dispatching rule when the stations were pooled and confirmed that this is a good choice. The underlying theoretical model

can be considered as a G/G/S queuing system at each machine group (i.e., general probability distribution of the inter-arrival and service times, with S servers in the machine group), in which a "customer" goes to the station that can first release him/her based on the service time and capability of the server.

The problem can be more complex than in Shaw's formulation if the manufacturing system rather resembles a multi-stage system in which each machine group is just one phase in the process (a possible model would be a network of G/G/S systems). In any case, if pooling is available we assume that the configuration remains fixed during the run time.

We assume that pallets and fixtures are available when a part is dispatched.

Different parts in the family need not have the same sequence nor they need to use all the stations. The total operative time of each station during the planning horizon (of hours or days) is shared by various part types. Since the production requirements of each part are assumed to be given by a higher planning level, say from MRP, all we can do is to check whether the time available is long enough to meet the production target. A lower bound for the run time can be easily obtained by using the expression:

$$\text{Min. Time Required} = DF \cdot \text{Max}_j \left[\frac{1}{M(i)} \sum_j^N P(i,j) R(j) \right] \quad (5.13)$$

with $P(i,j)$ being the process time part type j in machine type i ($i=1,\dots,M$; $j=1,\dots,N$), $R(j)$ the requirement of part type j , and $M(i)$ the number of machines type i . The coefficient DF is explained below.

Equation (5.13), however, does not determine the total processing time for the mix because of the precedence relations of the operations. Even for the time allocated to the bottle-neck station, equation (5.13) gives a lower bound, due to delays in transportation, setup, and eventual failures. Thus, a delay factor DF , ($DF \geq 1$) multiplying (5.13) would be needed to achieve better match with the actual time required (e.g., $DF = 1.25$).

Model (4.5) in section 4.7 can be expressed as:

$$\begin{aligned} & \text{Min } \sum_j^N c(j,t) \cdot u(j,t) & (5.14) \\ & \text{subject to } \sum_j DF \cdot P(i,j) \cdot u(j,t) \leq M(i,t) \quad (i=1,\dots,M) \\ & u \geq 0 \end{aligned}$$

Model (5.14) allows us to find the theoretical production of part type j by:

$$U(j,t) = \int_{s=0}^{s=t} u(j,s) \, ds \quad (5.15)$$

We need to develop rules in order for CM to dispatch parts so $U(j,t)$ is met. In the works by Gershwin et. al [5,71,72], a "near-continuous" flow of parts into a cell is assumed so that at each time step a part type j is dispatched when:

$$\begin{array}{rcl} \text{Actual buffer of part type } j & & \\ & \leq & (5.16) \\ \text{Theoretical buffer of part } j & & \end{array}$$

Where the buffer is the difference between the production level and demand. Rule (5.16) is appropriate for manufacturing systems such as high-speed transfer lines or flow shops (e.g., assembly of chips on computer boards) where the processing times are very small (order of seconds) and the output rate is high (order of thousands).

In our case, we attempt to meet the production rates by using production ratios, as introduced in section 4.5.2. From (5.14) above, the rates are guaranteed to be feasible (in terms of the capacity available during some interval). By handling the rates, we intend to find the proportion in which parts are fed into the cell. To illustrate this, consider the following example.

Example 5.3. A batch consists of two part types, P1 and P2. Both types use two stations, drilling (M1) and milling (M2). The relative proportion in the number of pieces of

each type is not affected by the sequence in which P1 and P2 are dispatched. The processing times, in minutes, are given in the first two columns of Figure 5.3. Suppose that at some point in time the estimated surplus of each part type is given by the value of $X(t)$. Furthermore, the desired safety level is H . Suppose the cost coefficients, $c(j,t)$, are given by $c(j,t) = [X(j,t) - H(j,t)]$.

By solving (5.14) with the delay factor DF set to one, the optimal rates are shown in vector u .^{*} (Note that both part types are below the target level). The exact ratio of parts type 1 over type 2 is 4/5. Thus, until the next recalculation of (5.14) parts can be loaded, for instance, in the sequence 111122222.

	M1	M2	X	H	C	[*] u
P1	10	5	2	5	-3	0.0500
P2	8	12	0	4	-4	0.0625

Proc. Times

Figure 5.3: Input Data and Production Rates in Example 5.3

In Example 5.3, note also that the parts visit both stations. If a failure occurs in either station, the rate of both parts is temporarily zero because the respective com-

ponent in the vector of resources becomes zero. Generally, only the dispatching of parts that include the failed station in their routing is preempted. When the failed station restarts, the coefficients $c(j,t)$ of the parts stopped will usually become negative. Depending on their magnitude, more capacity will be allocated to the part type with least coefficient. The increased allocation will be reflected in the new relative proportion in which parts are dispatched.

A simple heuristic to generalize from the example above is as follows:

Algorithm to get part type. Let Sequence be the current sequence of parts (e.g., 221111).

Main Procedure to Get Part Type: (5.17)

Step 0. Set Pass to 1.

Step 1. If Sequence is empty and Pass is 1 then

Calculate new rates, Get Sequence,
set Pass to 2, redo from Step 1.

Step 2. If Sequence is empty and Pass > 1 then return a

message for the cases below:

- (a) no more parts in batch,
 - (b) critical machine down,
 - (c) buffers above safety level,
- exit.

Else do Step 3.

Step 3. Return the first part type in Sequence for which there are remaining units in the batch.

Sub-procedure to get Sequence:

Step 1. Read the production rates calculated (5.18) from model (5.14).

Step 2. Divide rates by the minimum non-zero rate and round number (if all rates are zero do nothing).

Step 3. Append to the current sequence the result from Step 2.

5.5.2 Material Handling System

Carts transfer parts from the loading dock to the stations, from the stations to the unloading dock, and between any pair of stations. The request for carts originates when a part is loaded and thereafter by the routing requirements. An idle cart which is nearest to the requesting station is used. Carts may be near the completion of their current task (e.g., loading a buffer). So, generally, requests are to be served whenever the carts end their current task. If no cart can process the task, the request is queued until any free cart can process it. This queue is logical and it is managed by the CM for tracking purpose.

In order to minimize the effect of transportation delays, the cart selection and the path between two locations is

found by a shortest path algorithm implemented with the A* algorithm (seen in section 2.2.3).

The movement of carts and other handling devices is subordinated to the dispatching and routing policies discussed above. Thus, with the exception of path selection, the movement and actions upon arrival to a destination are handled in deterministic form. Appendix E shows the flow-chart to simulate the movements.

Because several carts move simultaneously, we imposed a discipline to avoid collisions. Any pair of logical units (stop points) defines a zone. Each zone is considered as a single server queuing system with service time equal to the travel time. One vehicle is allowed to travel on a zone at a time. A vehicle arriving to a busy zone stops at the beginning of the zone and seizes it when next zone becomes free (after the preceding vehicle has left).

The heuristic function to find the shortest path between two points, when a path must be assigned, doubles the cost (distance) of using a zone if this is blocked by a vehicle. This simple mechanism tends to avoid busy zones and will lead the search to bypass them. Davis [45] presents a study of modeling techniques for automated guided vehicles but the routes are predefined no matter what is the status of a zone. In fact, fixed routing is the norm unless the operator intervenes, looks the status of the traffic network, and

manually inputs the path. The AI algorithm allows "opportunistic" vehicle routing by using real-time information.

5.5.3 Random Factors

A number of random factors may affect a schedule. We are mostly interested in machine failures since this is the main factor that causes a decrease of capacity.

Failures times and subsequent repair times are randomly generated according to predefined distributions (e.g., exponential for failure times and uniform for repair times). Station failures may occur when a part is being processed, the part has been assigned to the station, or when it is being transferred. When a part is being processed and a failure occurs, the operation may or may not be resumed. The operator is asked to decide. This depends on whether the part in process is discarded as a result of the failure (e.g., a damaged tool or a failure in calibration could violate the tolerance of the part).

Depending on the operator's decision, the part may then be discarded, sent to another station if pooling exists, or be kept aside at the station to proceed with the repair.

5.6 An Overview of the Experiments Performed with CM

With the research methodology and software developed here and included in Appendix F, there could be an endless number of experiments to conduct. To satisfy the research object-

ives stated in Chapter 1, we are concerned with the following issues:

1. Demonstrate the use of the CM's rule-based language.
2. Test the effectiveness of the dispatching policy described in section 5.5.1 and compare the results with the theoretical production trajectories.
3. Determine factors affecting the equipment's utilization, flow times, and work in progress.
4. Test the effectiveness of the cart dispatching and routing policies in section 5.5.3.
5. Test effectiveness and responsiveness of CM to handle random equipment failures.
6. Verify the CM's flexibility to modify the knowledge base when variations in the decision rules and cell configuration are introduced.

CHAPTER 6

EXPERIMENTS AND RESULTS

In this chapter we conduct experiments with the CM to (a) verify the proper functioning of the CM and (b) to test its performance. In the first case we discuss a deterministic example showing the trace of a run. In the second case we extend the example to validate the knowledge base under random factors. We analyze how well the CM matches the optimal production trajectories and derive formulas explaining the behavior of selected performance indicators.

6.1 Deterministic Case

We assume that the cell operates without random failures at the workstations. In order to illustrate this case we run Example 6.1 with five part types, coded P01 through P05.

As explained in more detail in Appendix D, section D.6, the inputs to the CM are the objects' definition, the rule base, and an initialization file to setup the run. The outputs are the trace and a report file containing: actual and theoretical production trajectories, deviation between the two, station utilization, station queue size, and flow time of the parts.

Example 6.1. The object "part" of Figure 6.1 (a) shows the requirement for each part type. There are 35 parts in total, decomposed into 5 parts of type P01, 10 of P02, 5 of P03, 10 of P04, and 5 of P05. Suppose these belong to a family of cylindrical parts requiring operations on a DNC lathe (m1), milling center (m2), grinding center (m3), and boring center (m4).

The processing time of the operations (in minutes) and the machine group where they can be performed are shown under the object "operation" in Figure 6.1 (a). The operations for each part type are shown in the slot "operations" of the object "part."

We use the object "machine-group" to cover the more general case of several pooled machines (physically or logically) in each group. The slot "number-up" is maintained to keep track of the number of machines operating in each group. In this case, there is one machine per group and the number of running machines will be either zero or one. In general, the machine state changes when a failure occurs but in Example 6.1 we assume that all machines remain operational. Associated with each machine group there are workstations coded 3, 6, 10, and 13 which are the number (code) of the logical units.

To keep track of the production status, the object "part" is described by several slots. The slot "sequence" is to

```

(a)
(defobject operation
  (unit (is minutes))
  (delay-factor (is 1.25))
  (o11 (machine m1) (time 15)) (o12 (machine m2) (time 10))
  (o13 (machine m3) (time 20)) (o14 (machine m4) (time 10))
  (o21 (machine m1) (time 10)) (o22 (machine m2) (time 15))
  (o24 (machine m4) (time 10)) (o31 (machine m1) (time 10))
  (o33 (machine m3) (time 20)) (o34 (machine m2) (time 10))
  (o41 (machine m1) (time 10)) (o42 (machine m2) (time 20))
  (o43 (machine m3) (time 10)) (o44 (machine m4) (time 10))
  (o51 (machine m1) (time 10)) (o52 (machine m2) (time 15))
  (o53 (machine m3) (time 10)))
;;

(defobject machine-group
  (code (is m1 m2 m3 m4))
  (number-up (m1 1) (m2 1) (m3 1) (m4 1))
  (usage-ea (m1 0) (m2 0) (m3 0) (m4 0))
  (station (m1 3) (m2 6) (m3 10) (m4 13)))
;;

(defobject part
  (code (is p01 p02 p03 p04 p05))
  (sequence (is))
  (in-process (is p01 p02 p03 p04 p05))
  (operations (p01 (o11 o12 o13 o14))
               (p02 (o21 o22 o24))
               (p03 (o31 o33 o34))
               (p04 (o41 o42 o43 o44))
               (p05 (o51 o52 o53)))
  (requirement (p01 5) (p02 10) (p03 5) (p04 10) (p05 5))
  (demand-rate (p01 0) (p02 0) (p03 0) (p04 0) (p05 0))
  (t-completed (p01 0) (p02 0) (p03 0) (p04 0) (p05 0))
  (production-rate (p01 0) (p02 0) (p03 0) (p04 0) (p05 0))
  (safety-level (p01 0) (p02 0) (p03 0) (p04 0) (p05 0))
  (t-demand (p01 0) (p02 0) (p03 0) (p04 0) (p05 0))
  (t-production (p01 0) (p02 0) (p03 0) (p04 0) (p05 0))
  (t-buffer (p01 0) (p02 0) (p03 0) (p04 0) (p05 0))
  (production (p01 0) (p02 0) (p03 0) (p04 0) (p05 0))
  (buffer (p01 0) (p02 0) (p03 0) (p04 0) (p05 0)))
;;

(defobject cart
  (codes
   (are (c1 c2 c3 c4)))
  (speed
   (c1 200) (c2 200) (c3 200) (c4 200))
  (status
   (c1 stopped-idle) (c2 stopped-idle)
   (c3 stopped-idle) (c4 stopped-idle))
  (position
   (c1 15) (c2 15) (c3 15) (c4 15))
  (last-position
   (c1 15) (c2 15) (c3 15) (c4 15))
  (task
   (c1 stage) (c2 stage) (c3 stage) (c4 stage))
  (destination
   (c1) (c2) (c3) (c4))
  (load
   (c1) (c2) (c3) (c4))
  (path
   (c1) (c2) (c3) (c4)))
;;

(b)
(defobject station
  (current-load (3 3) (6 6) (10 10) (13 13))
  (status (3 idle) (6 idle) (10 idle) (13 idle))
  (time-down (3 0) (6 0) (10 0) (13 0))
  (time-failed (3) (6) (10) (13))
  (begin-current-op (3) (6) (10) (13))
  (test-end-current-op (3) (6) (10) (13))
  ;;

  (defobject lu
    (loading (is 1))
    (unloading (is 15))
    (staging (is 15))
    (next (1 2) (2 3 4) (3 4) (4 5 12) (5 6 7)
           (6 7) (7 8 9) (8 9) (9 10 11) (10 11)
           (11 12 5) (12 13 14) (13 14) (14 1 2 15) (15 11)))
    ;;

  (defobject lu-distance
    (1 (1 0))
    (2 (1 2 50))
    (3 (2 3 40) (4 50))
    (4 (3 4 40))
    (5 (4 50) (12 30))
    (6 (5 6 40) (7 50))
    (7 (6 7 40))
    (8 (7 8 50) (9 30))
    (9 (8 9 50))
    (10 (9 10 40) (11 50))
    (11 (10 11 40))
    (12 (11 12 50) (13 30))
    (13 (12 13 40) (14 50))
    (14 (13 14 40))
    (15 (14 150) (2 30) (15 50))
    (15 (1 50)))
    ;;

  (defobject zone-status
    (1 (2 free))
    (2 (3 free) (4 free))
    (3 (4 free))
    (4 (5 free) (12 free))
    (5 (6 free) (7 free))
    (6 (7 free))
    (7 (8 free) (9 free))
    (8 (9 free))
    (9 (10 free) (11 free))
    (10 (11 free))
    (11 (12 free) (5 free))
    (12 (13 free) (14 free))
    (13 (14 free))
    (14 (1 free) (2 free) (15 free))
    (15 (1 free)))
    ;;

```

Figure 6.1: Object Definition for Example 6.1.

```

(d)

(defobject zone-id
  (1 (2 z1-2))
  (2 (3 z2-3) (4 z2-4))
  (3 (4 z3-4))
  (4 (5 z4-5) (12 z4-12))
  (5 (6 z5-6) (7 z5-7))
  (6 (7 z6-7))
  (7 (8 z7-8) (9 z7-9))
  (8 (9 z8-9))
  (9 (10 z9-10) (11 z9-11))
  (10 (11 z10-11))
  (11 (12 z11-12) (5 z11-5))
  (12 (13 z12-13) (14 z12-14))
  (13 (14 z13-14))
  (14 (1 z14-1) (2 z14-2) (15 z14-15))
  (15 (1 z15-1) (15 z15-15)))
  ;;

(defqueue zone-queue
  (discipline fcfs)
  (z0-1) (z1-2) (z2-3) (z2-4) (z3-4)
  (z4-5) (z4-12) (z5-6) (z5-7) (z6-7)
  (z7-8) (z7-9) (z8-9) (z9-10) (z9-11)
  (z10-11) (z11-12) (z11-5) (z12-13) (z12-14)
  (z13-14) (z14-1) (z14-2) (z14-15) (z15-1)
  (z15-15))
  ;;

(defqueue buffer
  (discipline fcfs)
  (3) (6) (10) (13))
  ;;

(defqueue cart-assignment
  (discipline fcfs)
  (c1) (c2) (c3) (c4) (any))
  ;;

;; state variables

(setq *state-variables* t)

;; procedures

(defobject meta-rules
  (procedures (are move-cart cart-at)))
  ;;

```

```

(d)

(defobject time
  (status-step (is 10))
  (retry-load-step (is 5))
  (start (is 0))
  (end (is)))
  ;;

(defobject random-variable
  (load-delay (value (constant 1.0)))
  (unload-delay (value (constant 1.0)))
  (repair-time (value (uniform 10 20 *seed-1*) (mean 15)))
  (failure-time (value (exponential 1/240 *seed-0*) (mean 240)))
  ;;

  ;; statistics

(defstat s-t-production
  (type time-persistent)
  (show-values t)
  (p01) (p02) (p03) (p04) (p05))
  ;;

(defstat s-production
  (type time-persistent)
  (show-values t)
  (p01) (p02) (p03) (p04) (p05))
  ;;

(defstat tsys
  (type sample)
  (part))
  ;;

(defstat station-util
  (type time-persistent)
  (abstract (hget station time-down ?X))
  (3) (6) (10) (13))
  ;;

(defstat in-buffer
  (type time-persistent)
  (3) (6) (10) (13))
  ;;

```

Figure 6.1, continued.

store the dispatching sequence as explained in section 5.5.1. The demand rate is internally computed at any time t . In this case, the demand rate is a constant function equal to the ratio of requirements over the run time. The production rate is computed from the LP (5.14) and varies with time. The safety level is used in the computation of coefficients in the objective function of LP (5.14) (see Example 5.3); it is internally computed and was approximated as the mean demand during the repair time of the machines. The slots "t-demand," "t-production," and "t-buffer" are the predicted (theoretical) values of their actual counterparts. They are computed internally by using the "process-state" function in the library (in file LIBRARY.LSP of Appendix F). Recall that the buffer is production minus demand at a time t .

The remaining objects in Figure 6.1 (a) and 6.1 (b) describe the carts, the stations, logical units (LU) and their distances, in feet. The traffic zones are defined by any pair of LU's. Their status is tracked in the object "zone-status" and they are initially free but become blocked as the carts seize the zones, or get free as the carts leave. In Figure 6.1 (c) the object "zone-id" is a code to keep track of the queues at the zones. The queues are initialized with the "defqueue" macro and they are empty at time 0. Other queues are the buffers at each station and a logical queue to assign tasks to the carts.

Also in Figure 6.1 (c) there is a variable named `*state-variables*` which is set to true in order to activate the user-defined "process-state" function. The object "meta-rules" indicates which events are processed by procedural routines.

In Figure 6.1 (d) the object "time" has slots to specify intervals between loading requests (load-step), data collection (status-step) and between retries for loading (retry-load-step). The latter is used to perform periodical checks on the state variables when case (c) of algorithm (5.17) occurs. Recall that if the theoretical buffers are above the safety level the resulting production rates are zero. So, the periodical check is to wait long enough until the model determines when a new part must be dispatched.

Figure 6.1 (d) shows the definitions for the random variables (there are constants too, but they were included here for simplification). There is a definition for the repair time and time between failures, in minutes. Finally, Figure 6.1 (d) shows the specifications of statistics used in the example.

6.1.1 Trace of Sample Session

Taking Example 6.1 as reference, here we show a partial trace of a sample run and its results. As explained in section 5.4.2, events are continuously monitored and a

forward-chaining inference is performed to deduce the corresponding actions. All events, deductions, and the rules fired in each case can be traced. Here we include only some of these for illustration purposes (see Appendix D, section D.5, for further explanations about traces).

As shown in Figure 6.2 (a), at time 0, we specify the desired run time. In general, this value is unknown at this time but an ideal value can be provided. This run time estimate is given by the maximum usage (in time) required at the bottleneck station. In this example, the maximum occurs at machine group M2 and the value is 593.75 minutes. This number was computed with formula (5.13) with the delay factor $DF = 1.25$. As discussed in section 5.5.1, this factor corrects the machine usage to account for various delays not considered by the processing times.

When the first part P02-1 is loaded, an object "load" is created to trace the specific part within type P02 (Figure 6.2 (b)). The rule that fired in each case is shown in square brackets (e.g., [Load-Part0]). A sample of these rules is listed in Figure 6.3 (the complete listing is shown in Appendix C).

A cart is requested to move the part from the loading dock to station 3 (Fig. 1.1), which is the station selected to perform the first operation. Cart C1 is selected, the parameters are updated, and the action to move the cart is displayed.

Lisp> (load "cm")

Enter Knowledge-Base File To Load:kb
 Enter Objects Files To Load:floor
 Max. Usage Per Machine is 593.75 At Group M2
 What Is The Run Time :593.75

(a)

Cm> (xsim-all 0)

Time : 0
 Attn. : Load-Part
 Actions
 [Load-Part0] Response To Load-Part At 0 Is P02.
 [Load-Part01] Load-Part 15.
 [Load-Part1] Load P02-1 Type P02.
 [Load-Part1] Load P02-1 Rem-Operations (O21 O22 O24).
 [Load-Part1] Part P02-1 Loaded At 0 At 1 Assigned To
 Station 3.
 [Load-Part2] Station Status 3 Waiting-Load.
 [Load-Part2] Load P02-1 Next-Station 3.
 [Load-Part2] Load P02-1 Time-In 0.
 [Load-Part2] Cart-Request 0 At 1 For Load-In P02-1.

Time : 0
 Attn. : Cart-Request At 1 For Load-In P02-1
 Actions
 [Cart-Request1] Selected Cart Is (C1 (15 1)).
 [Cart-Request2] Cart Destination C1 1.
 [Cart-Request2] Cart Status C1 Moving-Assigned.
 [Cart-Request2] Cart Load C1 P02-1.
 [Cart-Request2] Cart Task C1 Load-In.
 [Cart-Request2] Cart Path C1 (1).
 [Cart-Request2] Move-Cart 0 C1 From 15 To 1.

(b)

Time : 2.7
 Attn. : Start-Process At 3 Of P02-1
 Actions
 [Start-Process1] Station Begin-Current-Op 3 2.7.
 [Start-Process1] Station Est-End-Current-Op 3 12.7.
 [Start-Process1] Station Status 3 Processing.
 [Start-Process1] Load P02-1 Rem-Operations (O22 O24).
 [Start-Process1] End-Process 12.7 At 3 Of P02-1.

(c)

Figure 6.2: Selected Traces in Example 6.1

Time : 12.7

Attn. : End-Process At 3 Of P02-1

Actions

[End-Process1] Station Begin-Current-Op 3 _.
 [End-Process1] Station Est-End-Current-Op 3 _.
 [End-Process1] Logical Unit 6 Selected At 12.7 For P02-1
 In 3.
 [End-Process2] Station Status 6 Waiting-Load.
 [End-Process2] Station Status 3 Waiting-Unload.
 [End-Process2] Cart-Request 12.7 At 3 For Unload-Machine
 [End-Process2] Load P02-1 Next-Station 6.

Attn. : Cart-Request At 3 For Unload-Machine _

Actions

[Cart-Request1] Selected Cart Is (C1 (3)).
 [Cart-Request2] Cart Destination C1 3.
 [Cart-Request2] Cart Status C1 Moving-Assigned.
 [Cart-Request2] Cart Load C1 _.
 [Cart-Request2] Cart Task C1 Unload-Machine.
 [Cart-Request2] Cart Path C1 Nil.
 [Cart-Request2] Move-Cart 12.7 C1 From 3 To 3.

(d)

Time : 13.7

Attn. : Free-Station At 3

Actions

[Free-Station2] Station Status 3 Idle.

(e)

Time : 15.35

Attn. : Start-Process At 6 Of P02-1

Actions

[Start-Process1] Station Begin-Current-Op 6 15.35.
 [Start-Process1] Station Est-End-Current-Op 6 30.35.
 [Start-Process1] Station Status 6 Processing.
 [Start-Process1] Load P02-1 Rem-Operations (024).
 [Start-Process1] End-Process 30.35 At 6 Of P02-1.

(f)

Figure 6.2, continued.

Time : 43.4

Attn. : End-Process At 13 Of P02-1

Actions

[End-Process1] Station Begin-Current-Op 13 _.
[End-Process1] Station Est-End-Current-Op 13 _.
[End-Process1] Logical Unit 15 Selected At 43.4 For P02-1
In 13.
[End-Process2] Part P02-1 Sent To Unloading Dock.
[End-Process2] Station Status 13 Waiting-Unload.
[End-Process2] Load P02-1 Next-Station 15.
[End-Process2] Cart-Request 43.4 At 13 For Unload-Machine
_.

(g)

Time : 44.4

Attn. : Free-Station At 13

Actions

[Free-Station2] Station Status 13 Idle.

(h)

Time : 45.85

Attn. : Unload-Part P02-1.

Actions :

[Unload-Part1] Part Production P01 1.
[Unload-Part1] Part %-Completed P02 10.

(i)

Time : 508.95

Attn. : End-Process At 13 Of P04-10

Actions

[End-Process1] Station Begin-Current-Op 13 _.
[End-Process1] Station Est-End-Current-Op 13 _.
[End-Process1] Logical Unit 15 Selected At 508.95 For
P04-10 In 13.
[End-Process2] Part P04-10 Sent To Unloading Dock.
[End-Process2] Station Status 13 Waiting-Unload.
[End-Process2] Load P04-10 Next-Station 15.
[End-Process2] Cart-Request 508.95 At 13 For
Unload-Machine _.

(j)

Figure 6.2, continued.

Time : 555
Attn. : Load-Part
Actions

[Load-Part0] Response To Load-Part At 555 Is
No-More-Parts-In-Batch.
[Load-Part3] Cannot Dispatch At Time 555.
[Load-Part3] All Parts Have Been Dispatched.

(k)

Time : 585.6
Attn. : End-Process At 13 Of P02-10
Actions

[End-Process1] Station Begin-Current-Op 13 _.
[End-Process1] Station Est-End-Current-Op 13 _.
[End-Process1] Logical Unit 15 Selected At 585.6 For
P02-10 In 13.
[End-Process2] Part P02-10 Sent To Unloading Dock.
[End-Process2] Station Status 13 Waiting-Unload.
[End-Process2] Load P02-10 Next-Station 15.
[End-Process2] Cart-Request 585.6 At 13 For Unload-Machine
_.

Time : 586.6
Attn. : Free-Station At 13
Actions

[Free-Station2] Station Status 13 Idle.

Time : 588.05
Attn. : Unload-Part P02-10
Actions

[Unload-Part1] Part Production P02 10.
[Unload-Part1] Part %-Completed P02 100.
[Unload-Part3] Check-End-Run.

(1)

Nil
Cm>

Figure 6.2, continued.

```

(defrule load-part0
  if
  (load-part ?tnow)
  then
  (response to load-part at ?tnow is (get-part-type)))
  ;;

(defrule load-part01
  if
  (load-part ?tnow)
  (response to load-part at ?tnow is ?ans
   (~ ?ans no-more-parts-in-batch))
  (time load-step is ?step)
  then
  ($ (load-part (+ ?tnow ?step)))
  ;;

(defrule load-part1
  if
  (response to load-part at ?tnow is ?type)
  (part in-process is ?type)
  (<- ?part (release-from-queue batch ?type)
   (has-value ?part))
  (part operations ?type ?ops)
  (in loading is ?dock)
  (<- ?station (get-next-machine ?ops))
  then
  (-> (load ?part type ?type
         ?part rem-operations ?ops))
  (part ?part loaded at ?tnow at ?dock assigned
   to station ?station))
  ;;

(defrule load-part2
  if
  (part ?part loaded at ?tnow at ?dock assigned
   to station ?station)
  then
  (if
   (station status ?station idle)
   then
   (-> (station status ?station waiting-load)))
  (if
   (station status ?station ?st (~ ?st idle))
   then (part ?part sent to buffer at station ?station))
  (-> (load ?part next-station ?station
         ?part time-in ?tnow))
  ($ (cart-request ?tnow at ?dock for load-in ?part))
  ;,exit))

```

```

(defrule load-part3
  if
  (response to load-part at ?tnow is
   no-more-parts-in-batch)
  then
  (cannot dispatch at time ?tnow)
  (all parts have been dispatched))
  ;;

(defrule cart-request1
  if
  (cart-request ?? at ?pos for ?? ??)
  then
  (selected cart is (nearest-cart ?pos)))
  ;;

(defrule cart-request2
  if
  (cart-request ?tnow at ?pos for ?task ?load)
  (selected cart is ?object (~ ?object none))
  (<- (?cart ?path) ?object)
  (cart status ?cart ?cstat
   (member ?cstat (stopped-idle)))
  (cart position ?cart ?cpos)
  then
  (-> (cart destination ?cart ?pos
        status ?cart moving-assigned
        load ?cart ?load
        task ?cart ?task
        path ?cart (rest-of ?path)))
  ($ (move-cart ?tnow ?cart from ?cpos to ?pos))
  (exit))
  ;;

(defrule start-process1
  if
  (start-process ?tnow at ?station of ?part)
  (load ?part rem-operations ?ops)
  (<- ?current-op (first ?ops))
  (operation ?current-op time ?ptime)
  then
  (if (station status ?station ?st (~ ?st down))
   then
   (-> (station begin-current-op ?station ?tnow
        est-end-current-op ?station (+ ?tnow ?ptime)
        status ?station processing))
   (-> (load ?part rem-operations (rest-of ?ops))
        ($ (end-process (+ ?tnow ?ptime) at ?station of ?part))))
  )

```

Figure 6.3 : Rules Fired in Example 6.1.

```

(start-process1 continued)

(if (station status ?station down)
    (cart position ?cart ?station))
then
    (station ?station is down part ?part is waiting)
    ($ (free-cart ?tnow ?cart at ?station))
    (-> (station est-rem-time ?station ?ptime))))
;;;

(defrule end-process1
  if
    (end-process ?tnow at ?station of ?part)
    (load ?part rem-operations ?ops)
    (<- ?next (get-next-machine ?ops))
  then
    (-> (station begin-current-op ?station
        est-end-current-op ?station _))
    (logical unit ?next selected at ?tnow for ?part in ?station))
  ;;;

(defrule end-process2
  if
    (logical unit ?next selected at ?tnow for ?part in
        ?station (== ?station ?next))
  then
    (if
        (station status ?next idle)
      then
        (-> (station status ?next waiting-load)))
      (if (station status ?next ?st (== ?st idle))
        then
          (part ?part sent to buffer at station ?next))
        (if (lu unloading is ?next)
          then
            (part ?part sent to unloading dock))
          (-> (station status ?station waiting-unload))
          (-> (load ?part next-station ?next))
          ($ (cart-request ?tnow at ?station
              for unload-machine _)))
        ...

(defrule free-station2
  if
    (free-station ?tnow at ?pos)
    (station status ?pos ?st (== ?st down))
    (queue buffer ?pos ?queue (has-no-value ?queue))
  then
    (-> (station status ?pos idle))
    (collect station-until ?pos ?tnow 0))
  ;;;

(defrule unload-part1
  if
    (unload-part ?tnow ?part)
    (load ?part type ?ptype)
    (load ?part time-in ?tin)
    (part production ?ptype ?n)
    (part t-production ?ptype ?tp)
    (part requirement ?ptype ?r)
    (<- ?prod (+ ?n 1))
  then
    (collect s-production ?ptype ?tnow ?prod)
    (collect s-t-production ?ptype ?tnow ?tp)
    (collect deviation ?ptype (- ?tp ?prod))
    (collect tsys part (- ?tnow ?tin))
    (-> (part production ?ptype ?prod
        $-completed ?ptype (/ (* 100 ?prod) ?r))))
  ;;;

(defrule unload-part2
  if
    (unload-part ?tnow ?part)
    (load ?part ?prop ?value)
  then
    (remove load ?part ?prop ?value))
  ;;;

(defrule unload-part3
  if
    (unload-part ?tnow ?part)
    (load ?part type ?ptype)
    (part requirement ?ptype ?r)
    (part production ?ptype ?n (>= (+ ?n 1) ?r))
  then
    (remove part in-process is ?ptype)
    (check-end-run))
  ;;;

```

Figure 6.3, continued.

The movement from LU 15 to LU 1 and then to LU 3 was omitted here by setting off the trace of movements. In subsequent experiments all traces are switched off just to obtain the results of the run.

At time 2.7 minutes the process of P02-1 begins (Figure 6.2 (c)). In Figure 6.2 (d), the process ends and a request to unload the station is made. Cart C1 was stopped and idle at LU 3 and when the request was made the same cart was selected since it was the nearest to that location. The part is moved to station at LU 6, the next station in the route. At time 15.35, the process starts at LU 6 (Figure 6.2 (f)).

Thus, parts are loaded and routed through the stations until completion. Figures 6.2 (g) and 6.2 (h) show when P02-1 finishes the processing at station 13 at time 43.4. The one minute delay before the station is free is due to the unload time. At time 45.85, P02-1 is unloaded from the cell (Figure 6.2 (i)).

The dispatching of parts continues until the entire batch has been loaded. Figure 6.2 (j) shows when the last part in the batch, P02-10, is ready to leave the first station.

In Figure 6.2 (k), the next part must be loaded but all parts have been already dispatched (there are still parts in process, however). Later at time 588.05, part P02-10 exits the cell and the run terminates (Figure 6.2 (l)).

Plots of the projected and actual production trajectory are shown in Figure 6.4, for the five part types. The observations are listed in Appendix G, section G.1.

The flat segments of the CM's trajectory correspond to the intervals in which the part type is not loaded. This is in response to the projected zero rates (as clearly seen in parts P01, P02, P03, and P05). If parts of a given type are in the cell, the flat segments represent the periods when the parts are in process.

Since the output of parts occurs at discrete times, the computation of the deviation between the model and the CM is meaningful only when the production is incremented by the unit (see rule "Unload-Part1"). Figure 6.5 shows the difference between the theoretical value obtained from expression (5.15), which is the integral of the production rates found by model (5.14), and the CM's values that resulted from algorithm (5.17). Since the deviation measures the closeness of the two trajectories, we use the mean value and the standard deviation as measures for validation. We expect the deviation to be small. Later, in section 6.2, we perform an experimental design to examine the sensitivity of deviation under different parameters for the distributions of the repair time and time between failures.

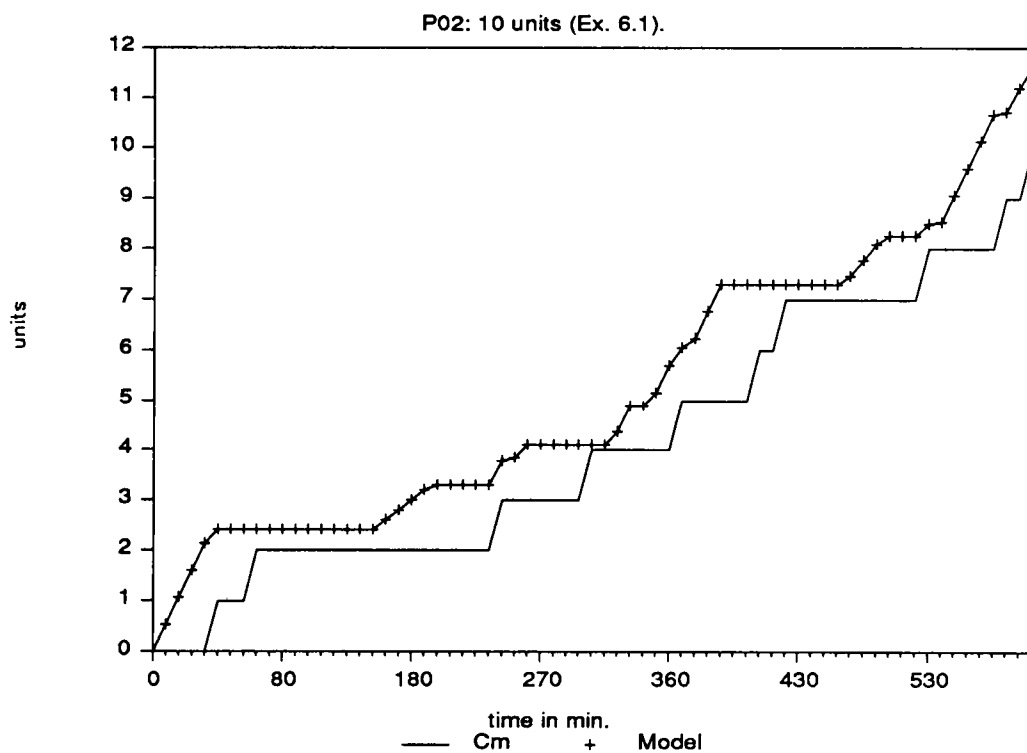
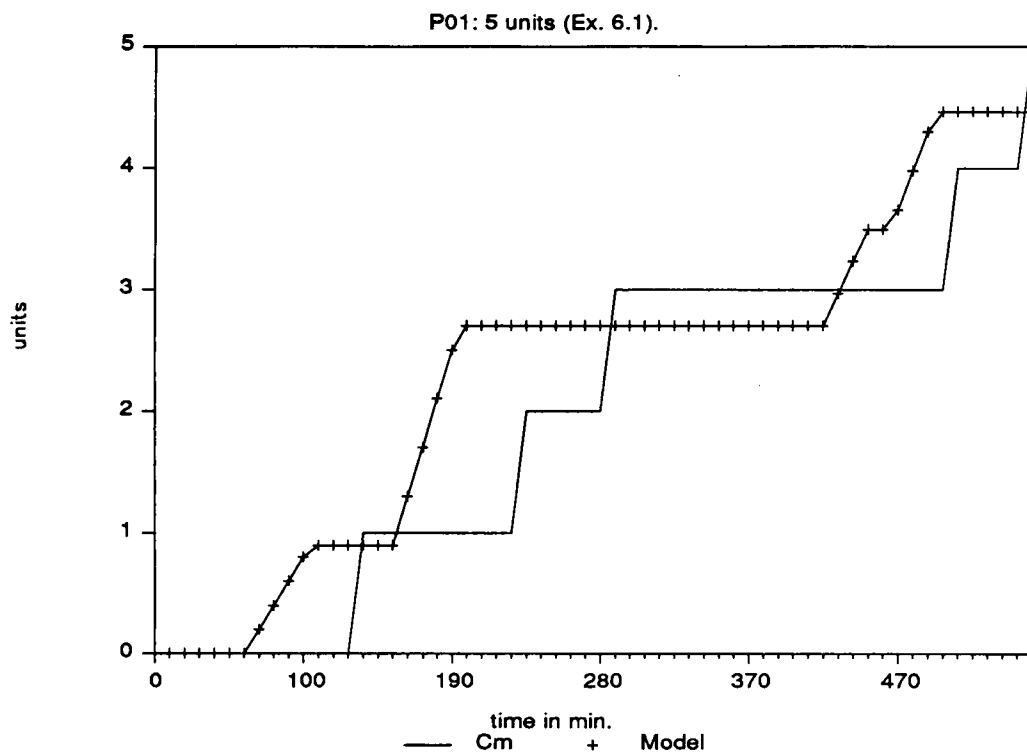


Figure 6.4: Production Trajectory of Parts in Example 6.1.

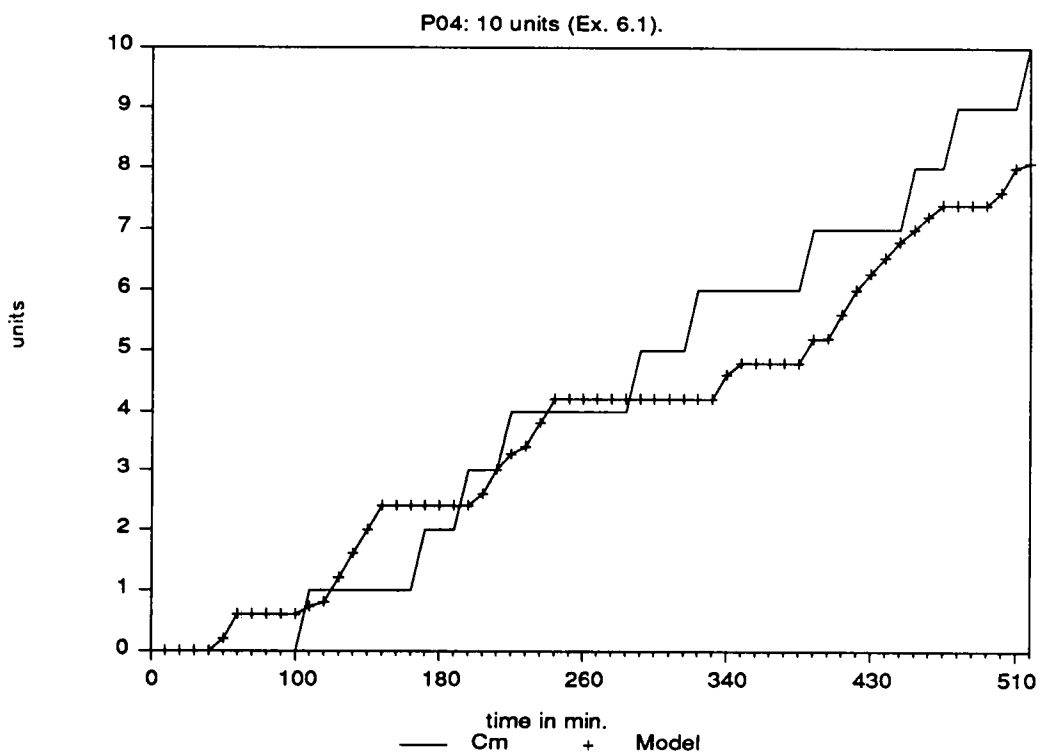
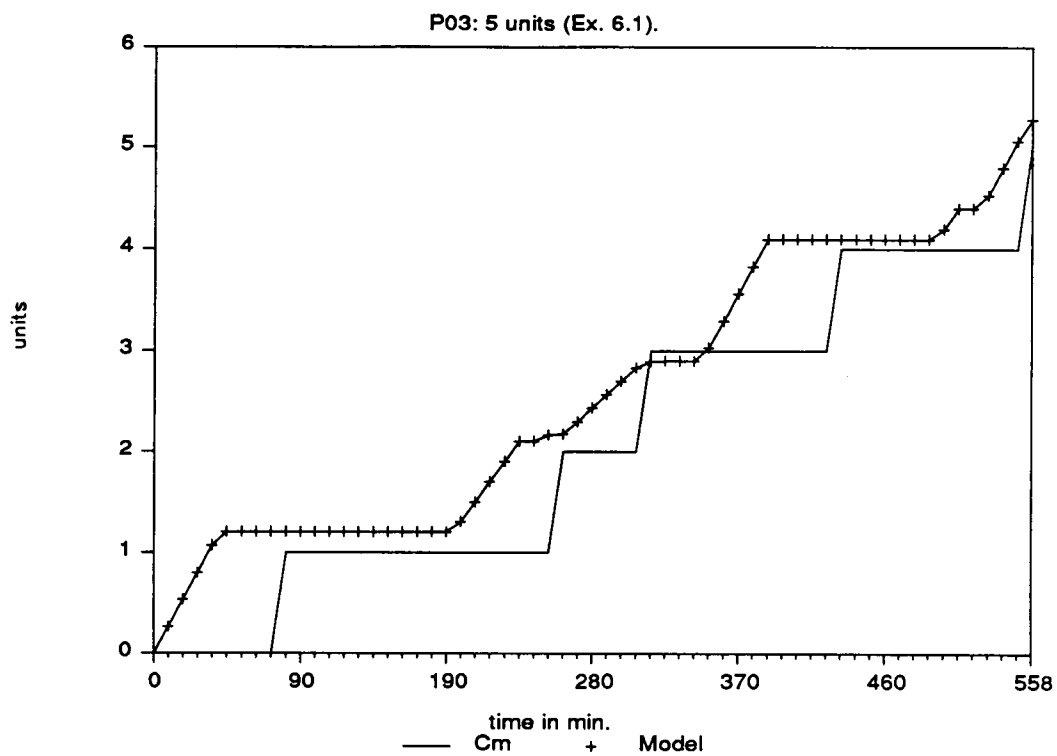


Figure 6.4, continued.

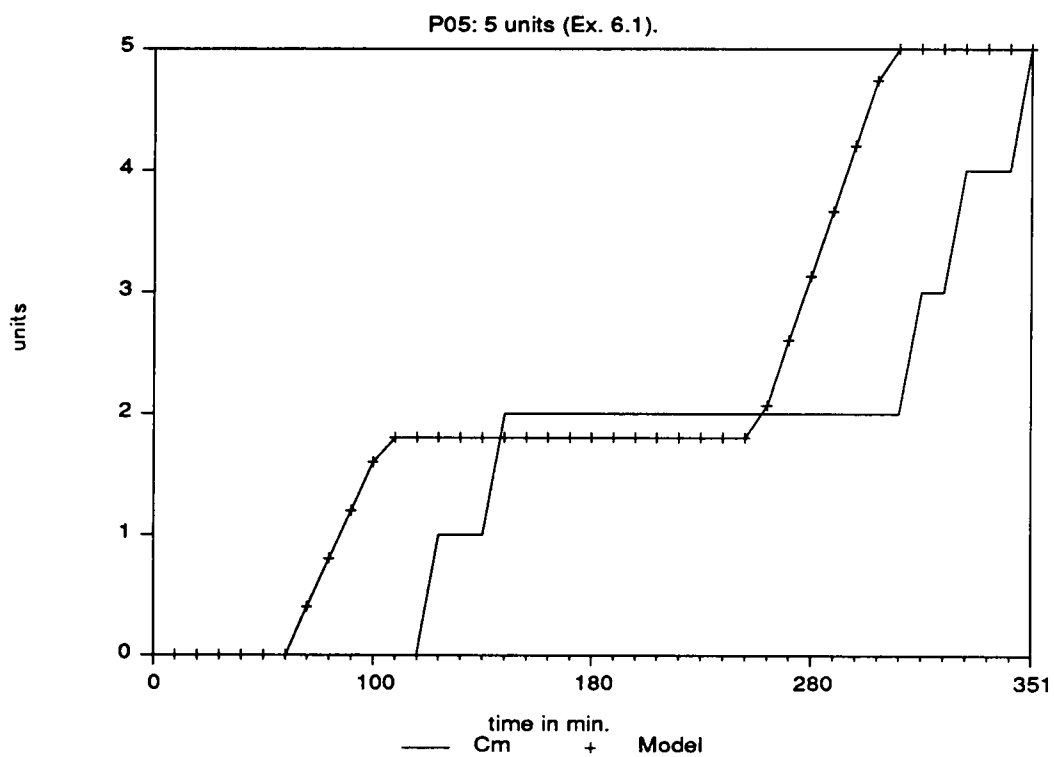


Figure 6.4, continued.

The deviation is attributable to the following factors:

(a) When rates are computed by model (5.14), the time-varying mix of parts is determined by algorithm (5.17) which obtains relative proportions in an integer number of parts.

(b) The processing times are in the order of minutes. Intuitively, as these times become smaller (order of seconds) the flat segments of the production curves are shorter and closer to the theoretical values.

(c) Each time a part type is loaded, the part is selected and the part itself is pulled from the storage system. Thus, there is a lag before the actual production starts responding to the theoretical rate.

(d) The delays in the material handling system, although small, are inevitable. This has been included in the factor $DF = 1.25$. Too large or too small (e.g., $DF = 1$) values cause an overestimate or underestimate of the run time. This translates into theoretical rates slower or faster than the actual throughput of the system. Fortunately, DF is only one parameter and it is simple to estimate.

The run time (needed to complete the batch) was 588.05. The estimated value was 593.75, including the delay factor $DF = 1.25$.

Figure 6.5 also shows the station utilization. This is a time-integrated average (Figure 4.3) computed from observations taken when the station is loaded with the part and when unloading ends (i.e., when the cart assigned to pick up the part leaves the station and goes to the next logical unit in the part's routing). Station 6 at group M2 has the highest utilization which is consistent with the maximum usage indicated at the beginning of the run (Figure 6.2 (a)).

The time-integrated average of the buffers at each station (In-Buffer) shows station 6 having the highest value (bottleneck machine).

Finally, the mean time in the system (flow-time) for all parts was 65.60 minutes with a STD of 15.90 minutes (shown under Tsys). This is a simple statistic and includes the time spent by the part since it was loaded at dock LU 1 and unloaded at dock LU 15.

Example 6.2. In this example we simply tested the effect of having more parts to manufacture. Production requirements were doubled by using:

```
(defobject part
  (requirement (p01 10) (p02 20) (p03 10)
               (p04 20) (p05 10)).
```

The run time was specified as 1187.5 minutes and the actual run ended at time 1090.10. The results are shown in Figures 6.6 and 6.7. The production trajectories follow the same trend up to the run time of Example 6.1.

Figure 6.7 shows that utilization increased few points and no major deterioration in the other indicators occurs. As in the results for Example 6.1 (Figure 6.5) note that the mean deviation of the theoretical and CM's production trajectory are reasonable small, as well as the standard deviations. As we include random factors, we want to examine if these values still remain stable.

Production was met within the estimated run time. The run time is estimated from the total processing time of the bottleneck machine group (M2). As known, the run time cannot be shorter than the time allocated at M2 but the margin (given by $DF = 1.25$) is built to include the time spent in transportation and loading/unloading at the stations. In Example 6.1, the actual run time was 588.05 min. (compared to the estimated 593.75 min.) whereas in Example 6.2 it was 1090.10 min. (compared to 1187.5 min.).

A final observation about these two examples is that the mean deviation between predicted and actual trajectory is small. This is important to guarantee the viability of the approach when including machine failures, as seen in the next sections.

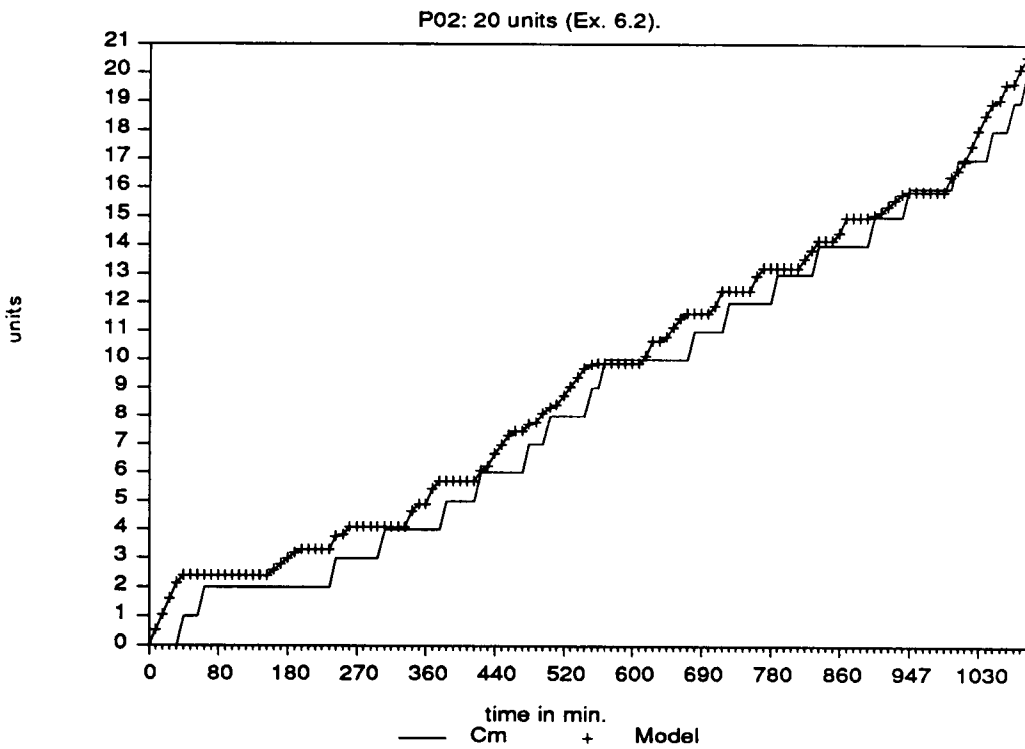
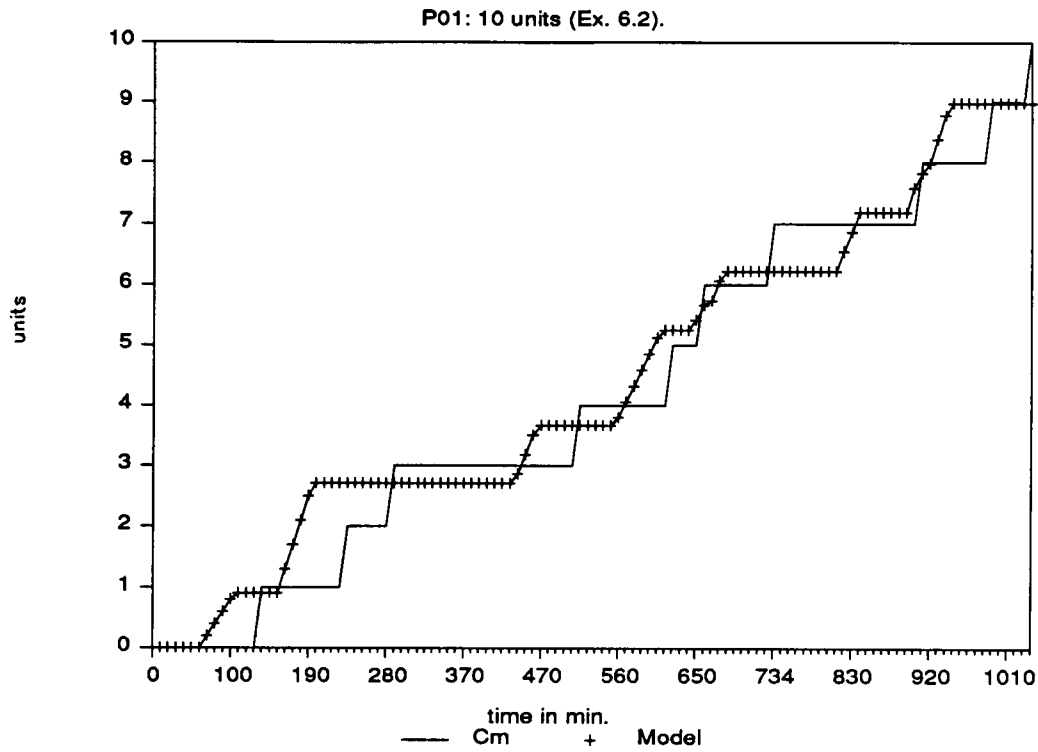


Figure 6.6: Production Trajectory of Parts in Example 6.2.

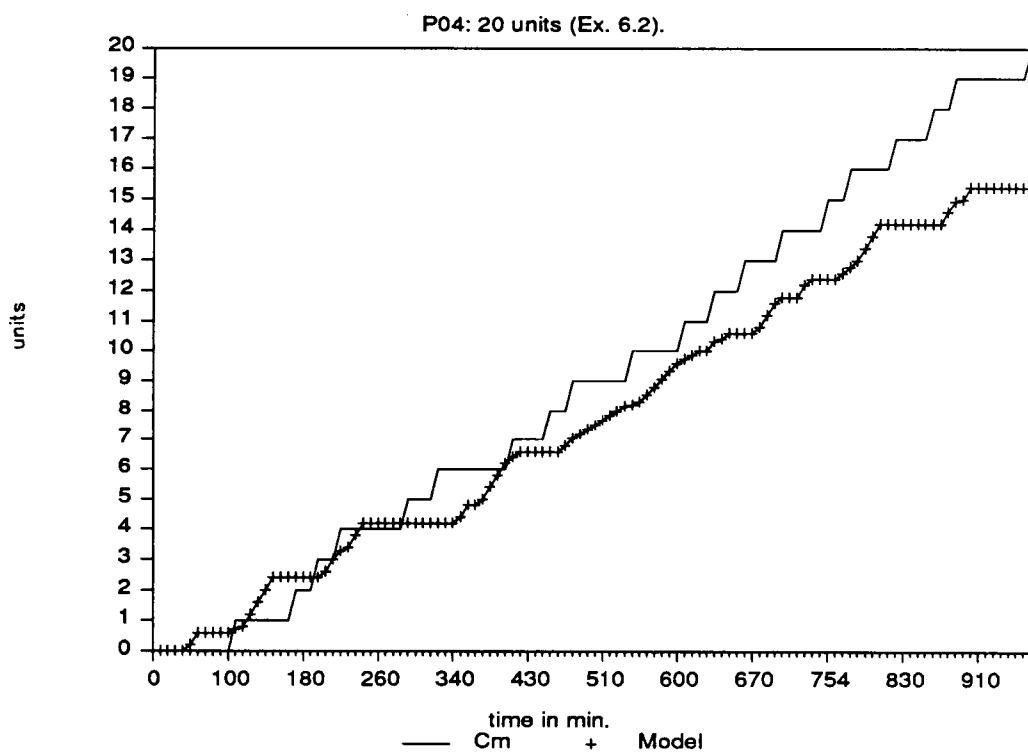
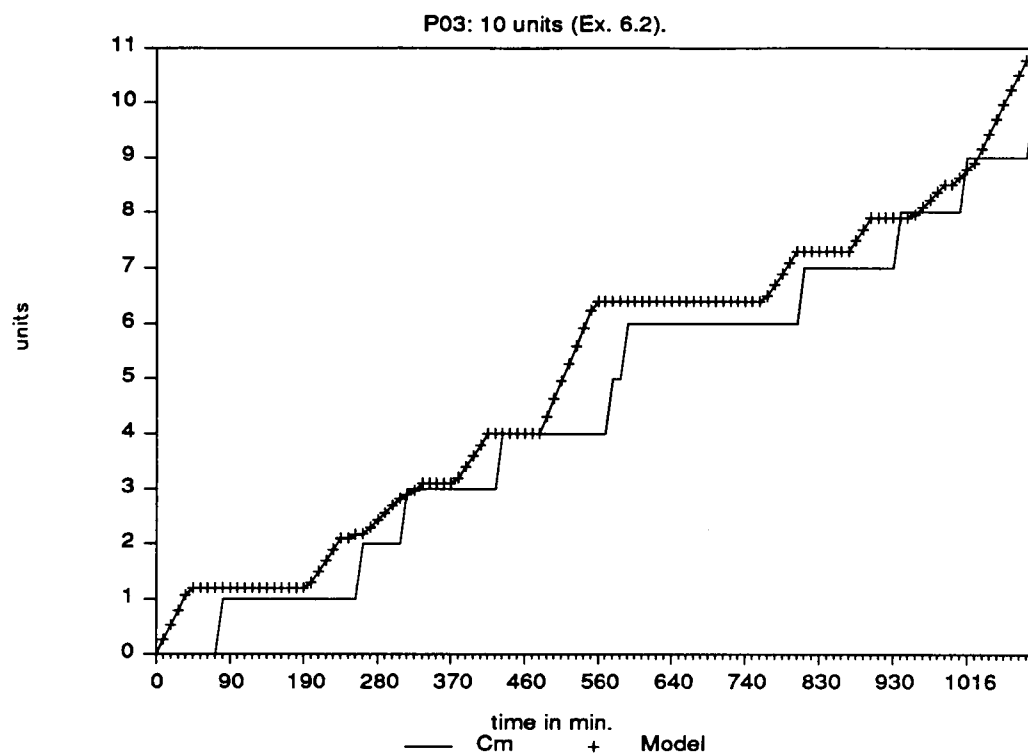


Figure 6.6, continued.

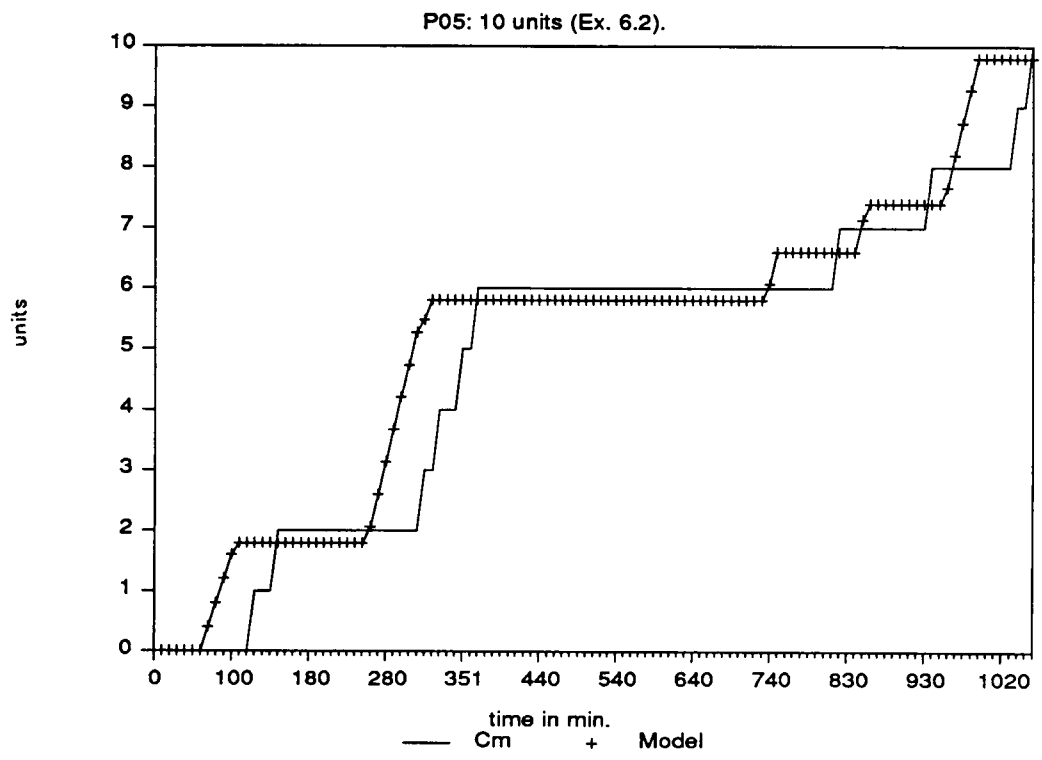


Figure 6.6, continued.

Report Example 6.2 Time = 1090.10

Deviation					
	P01	P02	P03	P04	P05
	-0.10	1.40	0.20	-0.28	0.80
	0.70	0.40	0.17	0.40	-0.20
	-0.30	0.78	-0.11	-0.60	2.48
	-0.34	0.10	0.00	-0.72	1.80
	0.26	0.70	1.40	-0.80	0.80
	-0.33	0.09	0.40	-1.80	-0.20
	-0.78	0.72	0.30	-0.57	-0.40
	-0.17	0.34	-0.10	-1.40	-0.60
	-0.02	0.82	-0.22	-1.94	0.80
	-1.02	-0.14	0.77	-1.79	-0.20
		0.62		-1.28	
		0.42		-1.67	
		0.22		-2.40	
		0.18		-2.20	
		0.07		-2.60	
		-0.12		-3.20	
		-0.38		-2.80	
		0.94		-3.80	
		0.65		-4.07	
		0.57		-4.60	
Avg.	-0.21	0.42	0.28	-1.91	0.51
Std. Dev	0.49	0.43	0.49	1.33	1.02

Station-Utilization				
Station	3	6	10	13
Avg.	0.83	0.94	0.73	0.66

In-Buffer				
Station	3	6	10	13
Avg.	0.02	0.75	0.27	0.13

Tsys	
Avg.	72.43
Std. Dev.	16.46

Figure 6.7: Summary of Results in Example 6.2.

6.2 Examination Under Random Machine Failures

In addition to the rules that manage the part dispatching and flow within the cell, the random cases tested here require that the system adapts to the actual capacity of the cell as the status of the stations changes. Thus, when a failure occurs we first update the station status and the number of operational stations in the group so the dispatching algorithm (5.17) loads those parts that can still be processed at other stations. There exist critical stations used by all the parts in the mix, in which case no part can be loaded until the station restarts (e.g. station 3 in Example 6.1). In some cases, the part's routing skips the station and parts of that type may be loaded (e.g., part P03 may be loaded even if station 6 fails).

Also, when a station fails, a part can be in process. Then a decision must be made to resume the process after the restart, or else to scrap the part if, for instance, the material was damaged or it is out of tolerance. Here, we can enter into a diagnosis to return with the proper action to follow. In our case, we let the user to respond. As discussed below, the answer is obtained by calling a procedure, which in turn may activate a diagnosis expert system. This system may be goal-driven with backward-chaining control.

Figure 6.8 shows the rules used in the random case. Rule "failure0" asserts the new number of machines running in the group to which the failed station belongs, asserts the new

```

(defrule failure0
  if
    (station-failure ?tnow ?station)
    (machine-group station ?group ?station)
    (machine-group number-up ?group ?n)
  then
    (-> (machine-group number-up ?group (- ?n 1)))
    (-> (station status ?station down
      time-failed ?station ?tnow))
    (collect station-util ?station ?tnow 0)
    (calculate-new-rates t)
  ;;
)

(defrule failure1
  if
    (station-failure ?tnow ?station)
    (machine-group station ?group ?station)
    (machine-group number-up ?group ?n (<= ?n 1))
  then
    (erase-from-sequence-parts-using ?group)
  ;;
)

(defrule failure2
  if
    (station-failure ?tnow ?station)
    (station status ?station processing)
    (station current-load ?station ?part)
    (station est-end-current-op ?station ?tend)
    (<= ?ans (user-question1 ?tnow ?station ?part))
  then
    (process of part ?part at station ?station suspended at time ?tnow)
    (answer to resume after repair is ?ans)
    (preempt-event end-process ?tend at ?station of ?part)
    (remove station est-end-current-op ?station ?tend))
  ;;
)

(defrule failure3
  if
    (station-failure ?tnow ?station)
    (answer to resume after repair is t)
    (station est-end-current-op ?station ?tend)
  then
    (-> (station est-rem-time ?station (- ?tend ?tnow)))
  ;;
)

(defrule failure4
  if
    (process of part ?part at station ?station suspended at time ?tnow)
    (answer to resume after repair is nil)
    (load ?part ?prop ?value)
  then
    (part ?part removed from ?station at time ?tnow)
    (remove load ?part ?prop ?value))
  ;;
)

(defrule failure5
  if
    (process of part ?? at station ?station suspended at time ??)
    (answer to resume after repair is nil)
  then
    (-> (station current-load ?station
      est-rem-time ?station _))
  ;;
)

(defrule failure6
  if
    (station-failure ?tnow ?station)
    (<- ?repair-time (get-random repair-time))
  then
    ($ (station-restart (+ ?tnow ?repair-time) ?station)))
  ;;
)

(defrule restart1
  if
    (station-restart ?tnow ?lu)
    (machine-group station ?g ?lu)
    (machine-group number-up ?g ?n)
  then
    (-> (machine-group number-up ?g (+ ?n 1)))
    (calculate-new-rates t)
  ;;
)

(defrule restart2
  if
    (station-restart ?tnow ?station)
    (station time-failed ?station ?tfailed)
    (station time-down ?station ?tdown)
  then
    (if
      (station current-load ?station ?part (== ?part _))
      (station est-rem-time ?station ?trem)
      then
        (-> (station est-end-current-op ?station (+ ?tnow ?trem)
          status ?station processing)
          (collect station-util ?station ?tnow 1)
          ($ (end-process (+ ?tnow ?trem) at ?station of ?part)))
      (if (station current-load ?station _))
      then
        (-> (station status ?station idle))
        ($ (free-station ?tnow at ?station)))
      (-> (station time-down ?station (+ ?tdown (- ?tnow ?tfailed)))
        ($ (station-failure (+ ?tnow (get-random failure-time)) ?station)))
      ;;
    )
  )

```

Figure 6.8 : Rules Used With Random Failures.

status as down, keeps track of the time when it failed (the total time down is deducted from the run time to calculate the actual utilization), collects the utilization, and calculates the new rates.

Rule "failure1" is to eliminate parts which are already in sequence (algorithm (5.17)). This preempts the loading of new parts which otherwise would go to the station buffers.

Rule "failure2" fires when the station is processing a part. The variable ?ans is instantiated from the value returned by the function "user-question1." This function (appendix D) simply asks whether the process is resumed or the parts is discarded. This rule also preempts the scheduled completion of the operation (event end-process).

Rules "failure3" through "failure5" manage the necessary updates and rule "failure6" schedules the station's restart after a random repair time is generated (see description in object random-variable of Figure 6.1).

On the restart, rule "restart1" updates the number of machines running and calculates the new production rates so the parts using the station can be loaded.

Rule "restart2" will either cause the system to resume operations on the part or will set the station free to accept new parts or parts held in the station's buffer while the station was down.

Example 6.3. With the same data in Example 6.1 we test a case having random failures at station 3. The time between failures (TBF) was chosen as exponential with mean 240 minutes and the repair time (RT) as uniform between 10 and 20 minutes. Here we trace only one run but we summarize the results from ten replications under different sequences of random generators. (TBF uses *seed-0* and RT uses *seed-1* obtained from saved random states of the Vax Lisp system and listed in the file SEED-SET.LSP of Appendix F.)

Figure 6.9 shows the trace for part P04-5 loaded at 210 min. At time 212.69 the process starts and the completion of the operation (O41) is scheduled at time 222.69 but a failure occurs at time 216.269 (Figure 6.9 (c)). Since the operator asserts that the operation can be resumed when the station is restarted, the scheduled completion time is preempted. In Figure 6.9 (d), a part must be loaded. Given the current production trajectories, part types P02, P03, P04, or P05 could be loaded (either from the scheduled sequence or after calculating a new sequence) but station 3 is down and the corresponding message is displayed.

During the elapsed time when station 3 is being repaired other events not shown here have occurred. Upon the restart of station 3 at time 232.8175, capacity is regenerated and the new rates are calculated (Figure 6.9 (e)). The processing of P04-5 is resumed and a new end-process event is scheduled at the end of the remaining machining time.

Time : 210
 Attn. : Load-Part
 Actions
 [Load-Part0] Response To Load-Part At 210 Is P04.
 [Load-Part01] Load-Part 225.
 [Load-Part1] Load P04-5 Type P04.
 [Load-Part1] Load P04-5 Rem-Operations (041 042 043 044).
 [Load-Part1] Part P04-5 Loaded At 210 At 1 Assigned To
 Station 3.
 [Load-Part2] Station Status 3 Waiting-Load.
 [Load-Part2] Load P04-5 Next-Station 3.
 [Load-Part2] Load P04-5 Time-In 210.
 [Load-Part2] Cart-Request 210 At 1 For Load-In P04-5.

(a)

Time : 212.6999
 Attn. : Start-Process At 3 Of P04-5
 Actions
 [Start-Process1] Station Begin-Current-Op 3 212.6999.
 [Start-Process1] Station Est-End-Current-Op 3 222.6999.
 [Start-Process1] Station Status 3 Processing.
 [Start-Process1] Load P04-5 Rem-Operations (042 043 044).
 [Start-Process1] End-Process 222.6999 At 3 Of P04-5.

(b)

Time : 216.269
 Attn. : Station-Failure 3
 Actions
 [Failure0] Machine-Group Number-Up M1 0.
 [Failure0] Station Status 3 Down.
 [Failure0] Station Time-Failed 3 216.269.
 [Failure0] Calculate-New-Rates.
 [Failure1] Erase-From-Sequence-Parts-Using M1.
 [Failure2] Process Of Part P04-5 At Station 3 Suspended At
 Time 216.269.

Can Process For Part P04-5 Be Resumed At Station 3 : yes

(c)

Figure 6.9: Selected Trace in Example 6.3.

(Fig. 6.9 (c), continued)

[Failure2] Answer To Resume After Repair Is T.
 [Failure2] Preempt-Event End-Process 222.6999 At 3 Of
 P04-5.
 [Failure3] Station Est-Rem-Time 3 6.430969.
 [Failure6] Station-Restart 232.8175 3.

(c)

Time : 225
 Attn. : Load-Part
 Actions
 [Load-Part0] Response To Load-Part At 225 Is
 Critical-Machine-Down.
 [Load-Part01] Load-Part 240.
 [Load-Part4] Part P02 Uses Group M1.
 [Load-Part4] No Machine Available In Group M1 At Time 225.
 [Load-Part4] Part P03 Uses Group M1.
 [Load-Part4] Part P04 Uses Group M1.
 [Load-Part4] Part P05 Uses Group M1.

(d)

Time : 232.8175
 Attn. : Station-Restart 3
 Actions
 [Restart1] Machine-Group Number-Up M1 1.
 [Restart1] Calculate-New-Rates.
 [Restart2] Station Est-End-Current-Op 3 239.2485.
 [Restart2] Station Status 3 Processing.
 [Restart2] End-Process 239.2485 At 3 Of P04-5.
 [Restart2] Station Time-Down 3 16.54852.
 [Restart2] Station-Failure 433.5881 3.

(e)

Time : 239.2485
 Attn. : End-Process At 3 Of P04-5
 Actions
 [End-Process1] Station Begin-Current-Op 3 _.
 [End-Process1] Station Est-End-Current-Op 3 _.
 [End-Process1] Logical Unit 6 Selected At 239.2485 For
 P04-5 In 3.

(f)

Figure 6.9, continued.

(Fig. 6.9 (f), continued)

```
[End-Process2] Station Status 6 Waiting-Load.
[End-Process2] Station Status 3 Waiting-Unload.
[End-Process2] Load P04-5 Next-Station 6.
[End-Process2] Cart-Request 239.2485 At 3 For
                Unload-Machine _.
```

(f)

```
Time      : 240
Attn.     : Load-Part
Actions
[Load-Part0] Response To Load-Part At 240 Is P03.
[Load-Part01] Load-Part 255.
[Load-Part1] Load P03-3 Type P03.
[Load-Part1] Load P03-3 Rem-Operations (O31 O33 O34).
[Load-Part1] Part P03-3 Loaded At 240 At 1 Assigned To
                Station 3.
[Load-Part2] Part P03-3 Sent To Buffer At Station 3.
[Load-Part2] Load P03-3 Next-Station 3.
[Load-Part2] Load P03-3 Time-In 240.
[Load-Part2] Cart-Request 240 At 1 For Load-In P03-3.
```

```
Time      : 240.9485
Attn.     : Free-Station At 3
Actions
[Free-Station2] Station Status 3 Idle.
```

(g)

```
Time      : 408.7
Attn.     : Free-Station At 3
Actions
[Free-Station1] Release-From-Queue Buffer 3.
[Free-Station1] Station Current-Load 3 P05-4.
[Free-Station1] Start-Process 408.7 At 3 Of P05-4.
```

(h)

Figure 6.9, continued.

The process ends at time 239.2485, the next station selected is station 6, and a cart request to unload the part is made (Figure 6.9 (f)).

In this case, when station 3 became free there were no parts in the buffer and the station was set idle (Figure 6.9 (g)). In other instances, as shown in Figure 6.9 (h), the next part in the buffer is released.

Plots of the projected and actual production trajectory for this particular run are shown in Figure 6.10.

Figure 6.11 presents the summary report of ten different runs. As compared to the results of Example 6.1 (in Figure 6.5), the run time increased from 588.05 min. to an average of 622.76 min. The mean flow time (T_{sys}) increased from 65.90 min. to 66.84 min. The mean queue sizes at the stations (In-Buffer) did not deteriorate and they remain close to those in Example 6.1. This can be explained because when failures occur, the loading of parts using unavailable stations is preempted and material does not accumulate.

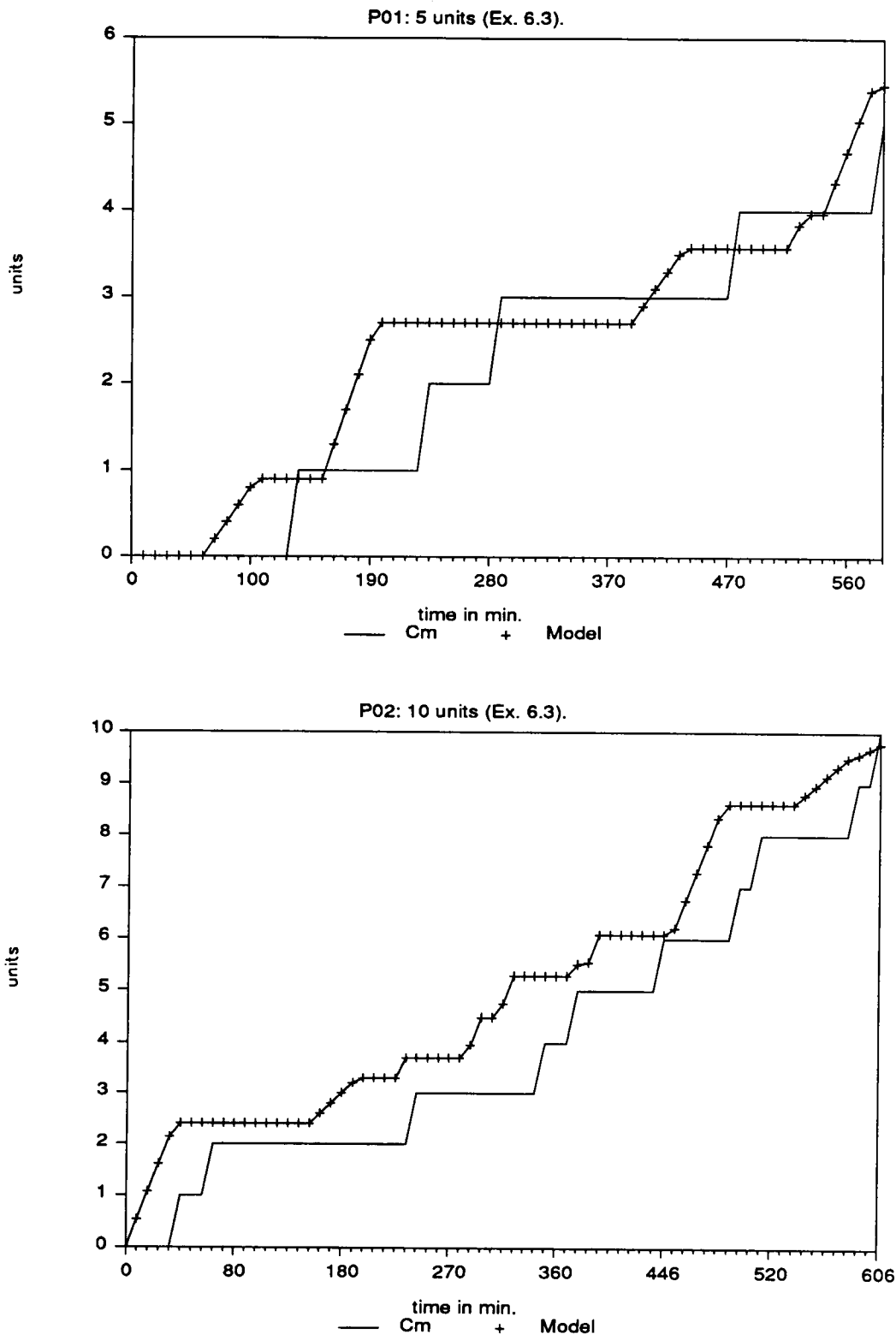


Figure 6.10: Production Trajectory of Parts in Example 6.3.

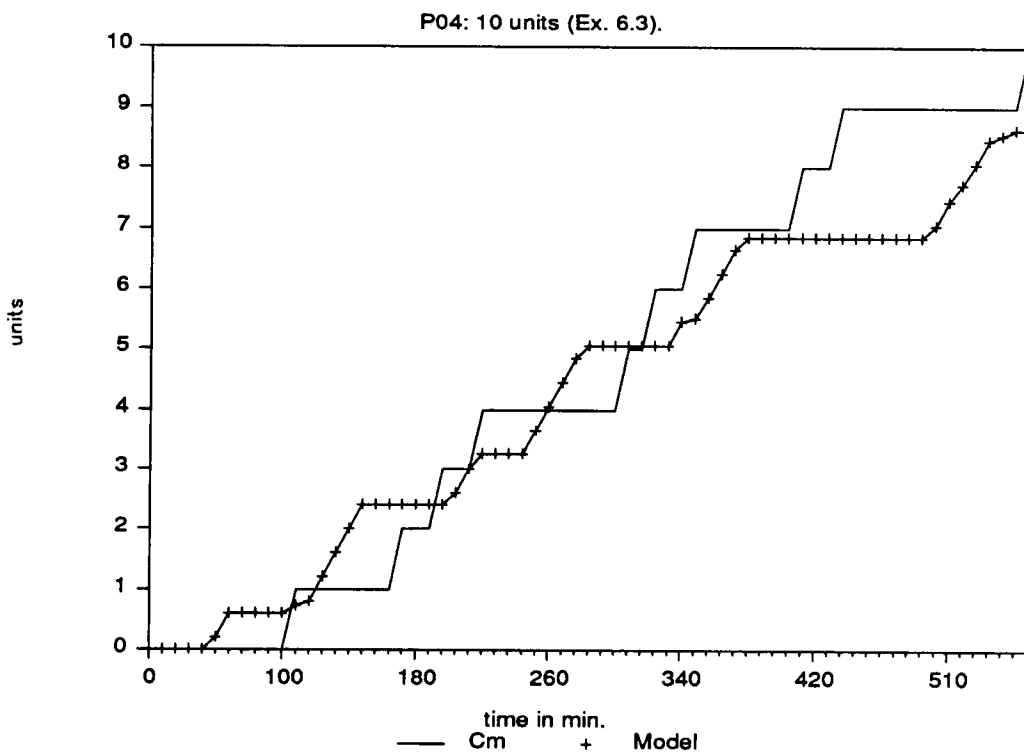
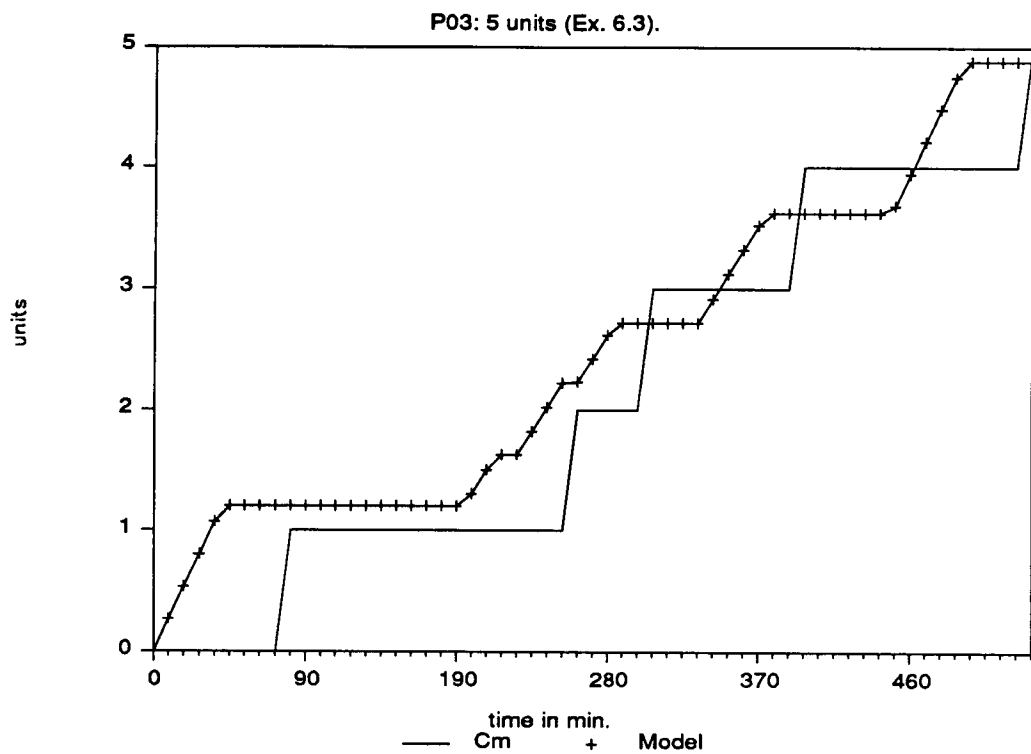


Figure 6.10, continued.

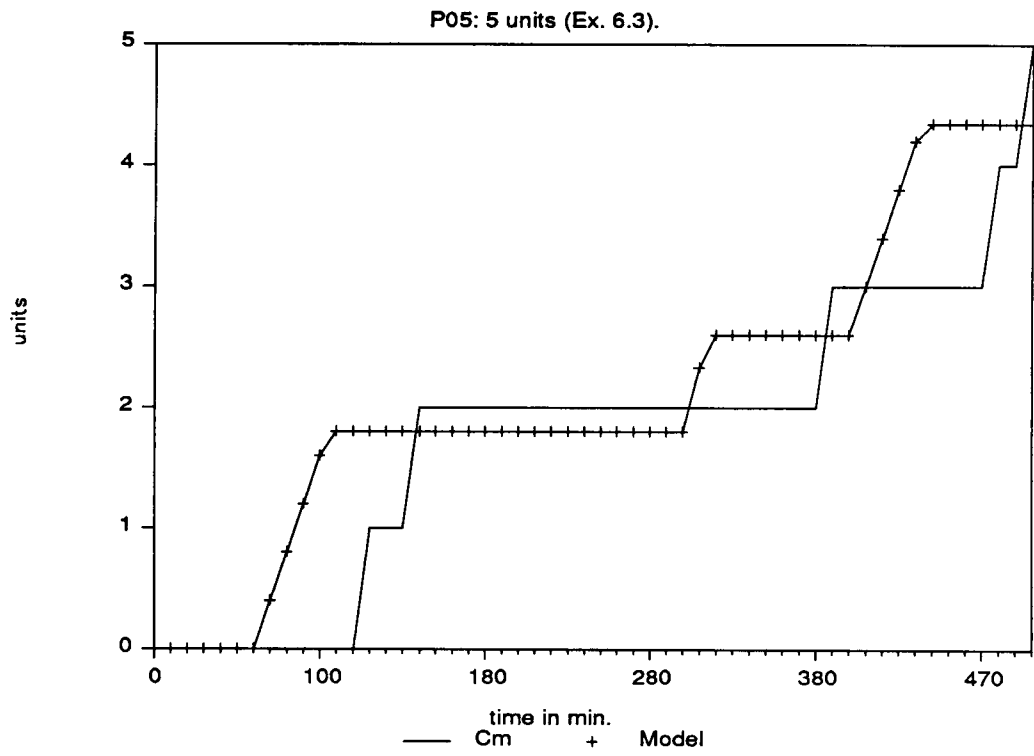


Figure 6.10, continued.

Summary Report Example 6.3 *

Mean Deviation					
Run	P01	P02	P03	P04	P05
1	-0.00	1.26	-0.27	-1.01	0.20
2	0.04	0.92	0.13	-1.01	0.72
3	-0.19	1.29	0.56	-0.71	0.20
4	-0.32	0.92	0.88	-0.06	-0.27
5	-0.30	0.93	0.13	-1.07	0.72
6	-0.10	1.00	-0.00	-0.69	0.00
7	0.02	0.86	0.03	0.20	-0.10
8	-0.49	0.71	-0.02	-0.60	0.87
9	-0.11	0.52	0.53	-1.03	0.72
10	-0.47	-0.43	1.02	0.56	0.11
Average	-0.19	0.80	0.30	-0.54	0.32
Std. Dev.	0.21	0.51	0.45	0.61	0.43

Station-Utilization					
Run	3	6	10	13	Run-Time
1	0.78	0.86	0.66	0.59	606.30
2	0.76	0.89	0.68	0.61	588.05
3	0.73	0.85	0.65	0.58	622.45
4	0.78	0.82	0.63	0.57	637.45
5	0.74	0.85	0.65	0.58	615.85
6	0.75	0.83	0.63	0.57	630.85
7	0.78	0.85	0.64	0.58	621.45
8	0.60	0.60	0.60	0.60	618.45
9	0.75	0.82	0.63	0.56	635.85
10	0.73	0.80	0.61	0.56	650.85
Average	0.74	0.82	0.64	0.58	622.76
Std. Dev.	0.06	0.08	0.02	0.02	18.54

* From 10 replications with RT ~ uniform(10,20) and
 TBF ~ exponential(1/240). TBF uses random state *seed-0*
 and RT uses *seed-1*

Figure 6.11: Summary of Results in Example 6.3.

In-Buffer Run	3	6	10	13
1	0.02	0.42	0.19	0.08
2	0.02	0.36	0.23	0.08
3	0.02	0.47	0.09	0.06
4	0.02	0.39	0.14	0.07
5	0.06	0.33	0.21	0.10
6	0.02	0.40	0.21	0.08
7	0.04	0.41	0.17	0.15
8	0.10	0.28	0.19	0.14
9	0.02	0.30	0.19	0.07
10	0.02	0.31	0.10	0.08
Average	0.03	0.37	0.17	0.09
Std. Dev.	0.03	0.06	0.05	0.03

Run	Tsys	Std. Dev. of Tsys
1	67.49	14.32
2	65.60	15.90
3	66.08	13.07
4	67.86	13.62
5	66.66	15.83
6	67.30	14.52
7	69.73	13.80
8	67.35	17.43
9	65.41	15.43
10	64.86	9.77
Average	66.84	
Std. Dev.	1.51	

Figure 6.11, continued.

6.3 Analysis of Results

In order to further investigate the effect of random disruptions over the CM's performance, we took Example 6.3 and run the CM under different levels of MTR and MTBF. We assume that all the stations fail according to the same exponential distribution.

For each combination level, five replications under different random states were made. In this section we explore the sensitivity of (a) the mean deviation between predicted and actual production trajectory, (b) the station utilization, (b) the station buffers, (d) the run time, and (e) the time the parts spend in the system.

For (a), we conduct an analysis of variance (ANOVA) over the mean deviation of the trajectory of each part type. We follow this approach because it is not possible to obtain a closed-form expression to measure the fitness of the CM's trajectory to the predicted trajectory. The heuristics discussed in sections 5.5.1 and 5.5.2 are intended to produce a good match between predicted and actual values but we need to test this. We are interested in showing that the deviation is independent of the parameters of the random variables RT and TBF, or that their effects are small. By small effect, we mean a close-to-zero mean and narrow confidence intervals.

For indicators (b) through (e) we discuss the results and later derive approximate formulas that explain the different trends.

We still assume that the TBF follows an exponential distribution and the RT has a uniform distribution. In the approximate formulas, we work with the mean values and the restriction on the distributions is eliminated.

We must suspect that some of the indicators (b) through (e) should suffer a deterioration as either failures are more frequent or the repair time is longer. The problem is, however, to quantify these losses.

6.3.1 Analysis for the Mean Deviation

In order to test the variation of the mean deviation, the distributions of the repair time (RT) and time between failures (TBF) were set to the levels shown in Table 6.1.

By running five replications under each combination, we obtained values for the mean deviation between predicted and actual production trajectories. These are labeled D1, D2, D3, D4, and D5 and correspond to the five part types.

D1-D5 are the response variables and an ANOVA is performed for each, with the levels shown in Table 6.1. To simplify notation, the levels are identified by the mean values of the distributions (MRT and MTBF).

D1-D5 are assumed to be distributed normally. In Appendix G, section G.5, we tested the normality of the residuals for

TABLE 6.1

PARAMETER LEVELS FOR EXPERIMENTAL DESIGN

REPAIR TIME (RT)			TIME BETWEEN FAILURES (TBF)
Uniform			Exponential
Lower	Upper	Mean (MTR)	Mean (MTBF) *
10	20	15	60
20	40	30	120
50	70	60	240
110	130	120	480

* All times in minutes.

each ANOVA at a significant level $\alpha = 0.05$. Both, the hypothesis of normality for the distribution (using Chi-Square test) and then the hypothesis that the mean is zero were not rejected at $\alpha = 0.05$. This indicates the adequacy of ANOVA and its assumptions.

Full ANOVA tables are shown in Appendix G, section G.5. Here, we summarize the results.

Table 6.2 shows that in most cases neither the distribution of RT nor TBF's affect D1-D5 (based on the Prob ($> F$), at $\alpha=0.05$). There are two exceptions: D1 and D2 were affected by factor MTBF (the mean value increase as MTBF becomes larger). Tables 6.3 and 6.4 show 95-percent confidence intervals for the various levels. The two cases, D1 and D2, affected by MTBF exhibit (Table 6.4) a greater

TABLE 6.2

EFFECT OF DISTRIBUTION PARAMETERS ON MEAN
DEVIATIONS

Variable	Factor	Prob (> F)	Significant * at $\alpha = 0.05$
D1	MRT	.3762	NO
D1	MTBF	.0005	YES
D1	INTERACTION	.3272	NO
D2	MRT	.8599	NO
D2	MTBF	.0000	YES
D2	INTERACTION	.0574	NO
D3	MRT	.3901	NO
D3	MTBF	.4008	NO
D3	INTERACTION	.2829	NO
D4	MRT	.7160	NO
D4	MTBF	.0515	NO
D4	INTERACTION	.3788	NO
D5	MRT	.2843	NO
D5	MTBF	.3504	NO
D5	INTERACTION	.6099	NO

* Source: ANOVA tables in Appendix G.

TABLE 6.3

CONFIDENCE INTERVALS FOR MEAN DEVIATIONS
GIVEN MRT LEVELS

Var.	MRT Lvl.	Sample Size	Level *	Standard Error	Lower ** Limit	Upper Limit
D1	15	20	-.8095	.1056	-1.0208	-.5982
D1	30	20	-.5725	.1056	-.7838	-.3612
D1	60	20	-.7880	.1056	-.9993	-.5767
D1	120	15	-.6360	.1219	-.8799	-.3921
D2	15	20	-.0110	.1743	-.3597	.3377
D2	30	20	.0535	.1743	-.2952	.4022
D2	60	20	.1230	.1743	-.2257	.4717
D2	120	15	.4686	.2013	.0660	.8714
D3	15	20	.0140	.1452	-.2766	.3046
D3	30	20	.0810	.1452	-.2096	.3716
D3	60	20	.2740	.1452	-.0166	.5646
D3	120	15	.3653	.1677	.0298	.7009
D4	15	20	.5940	.2167	.1606	1.0274
D4	30	20	.3685	.2167	-.0649	.8019
D4	60	20	.4860	.2167	.0526	.9194
D4	120	15	.1593	.2502	-.3412	.6598
D5	15	20	.2120	.1567	-.1014	.5254
D5	30	20	.2790	.1567	-.0344	.5924
D5	60	20	.2610	.1567	-.0524	.5744
D5	120	15	.7380	.1567	.3761	1.0999

* From ANOVA tables in Appendix G.

** 95% confidence interval.

TABLE 6.4

CONFIDENCE INTERVALS FOR MEAN DEVIATIONS
GIVEN MTBF LEVELS

Var.	MTBF Lvl.	Sample Size	Level *	Standard Error	Lower ** Limit	Upper Limit
D1	60	15	-1.1026	.1219	-1.3466	-.8587
D1	120	20	-.7570	.1056	-.9683	-.5457
D1	240	20	-.6885	.1056	-.8998	-.4772
D1	480	20	-.3745	.1056	-.5858	-.1632
D2	60	15	-.7126	.2013	-1.1154	-.3100
D2	120	20	-.0380	.1743	-.3867	.3107
D2	240	20	.3710	.1743	-.0223	.7197
D2	480	20	.7185	.1743	-.3698	1.0672
D3	60	15	.0673	.1677	-.2682	.4029
D3	120	20	.0070	.1452	-.2836	.2976
D3	240	20	.3455	.1452	.0549	.6361
D3	480	20	.2400	.1452	-.0506	.5306
D4	60	15	.6693	.2502	.1688	1.1698
D4	120	20	.5930	.2167	.1596	1.0264
D4	240	20	.6010	.2167	.1676	1.0344
D4	480	20	-.1280	.2167	-.5614	.3054
D5	60	15	.0086	.1809	-.3532	.3706
D5	120	20	.3965	.1567	.0831	.7099
D5	240	20	.3475	.1567	.0341	.6609
D5	480	20	.5555	.1567	.2416	.8684

* From ANOVA tables in Appendix G.

** 95% confidence interval.

variability at level $MTBF = 60$. That is, when all the stations fail at a rate of one failure per hour, which is the worst case tested here for this factor. However, the means of $D1$ and $D2$ are still small.

Given the intervals shown in Tables 6.3 and 6.4, we can observe that the closeness of the predicted and actual production trajectories is adequate.

Table 6.5 shows the average standard deviations (of $D1-D5$) obtained from each of the levels tested. Since this statistic does not behave as normal, we only observe that their values are small.

As a conclusion, the match between the predicted and actual trajectories is not significantly (95 percent) affected by the random factors examined here, and the mean deviations are small (order of one part, at worst). Of course, this applies to the example examined here, but since the matching algorithm in (5.17) is general and does not impose restrictions on the processing times, we expect results similar to those found in this example.

Based on the experiment performed in this section, we obtained average values for the utilization, buffers, run time, and time in the system. The results are shown in Tables 6.6 and 6.7. We examine these indicators next.

TABLE 6.5

AVERAGE STANDARD DEVIATION GIVEN
LEVELS OF MRT AND MTBF

MRT	MTBF	SDD1 *	SDD2	SDD3	SDD4	SDD5
15	480	0.48	0.69	0.53	0.71	0.76
30	480	0.65	0.67	0.35	0.81	0.57
60	480	0.54	0.73	0.43	0.77	0.77
120	480	0.58	0.79	0.49	0.78	1.04
15	240	0.60	0.68	0.48	0.85	0.76
30	240	0.57	0.79	0.47	0.84	0.74
60	240	0.66	0.82	0.67	0.98	0.80
120	240	0.76	0.78	0.56	0.93	0.85
15	120	0.47	0.84	0.62	0.71	0.63
30	120	0.67	0.73	0.28	1.00	0.65
60	120	1.00	1.00	0.71	1.03	0.77
120	120	0.77	0.86	0.85	0.89	0.90
15	60	0.76	0.97	0.72	0.90	0.48
30	60	0.90	1.11	0.59	1.04	0.61
60	60	0.90	0.90	0.85	1.01	0.64

* Based on S^2_{n-1} full table in Appendix G.

TABLE 6.6

AVERAGE STATION UTILIZATION AND RUN-TIME
GIVEN LEVELS OF MRT AND MTBF

MRT	MTBF	U3	U6	U10	U13	RUN-TIME
15	480	0.70	0.81	0.63	0.56	651.77
15	240	0.69	0.81	0.58	0.58	668.13
15	120	0.64	0.76	0.54	0.52	741.12
15	60	0.56	0.61	0.42	0.39	978.33
30	480	0.68	0.83	0.60	0.53	683.10
30	240	0.64	0.69	0.48	0.49	772.48
30	120	0.54	0.61	0.42	0.32	1038.22
30	60	0.44	0.49	0.32	0.25	1542.62
60	480	0.64	0.75	0.59	0.44	771.47
60	240	0.49	0.58	0.43	0.40	1079.31
60	120	0.28	0.33	0.23	0.19	2360.00
60	60	0.17	0.19	0.11	0.09	5251.18
120	480	0.54	0.59	0.44	0.44	1017.74
120	240	0.33	0.35	0.27	0.23	2062.40
120	120	0.23	0.25	0.19	0.14	3995.57

Source: Appendix G.

TABLE 6.7

AVERAGE BUFFER SIZE AND TIME IN SYSTEM
GIVEN LEVELS OF MRT AND MTBF

MRT	MTBF	B3	B6	B10	B13	TSYS
15	480	0.02	0.36	0.15	0.11	67.34
15	240	0.02	0.37	0.20	0.18	71.75
15	120	0.06	0.39	0.15	0.16	75.44
15	60	0.03	0.32	0.21	0.13	86.71
30	480	0.02	0.42	0.31	0.21	76.81
30	240	0.02	0.34	0.19	0.20	75.52
30	120	0.05	0.22	0.15	0.21	83.14
30	60	0.04	0.30	0.18	0.17	102.19
60	480	0.03	0.45	0.33	0.29	84.20
60	240	0.04	0.30	0.27	0.31	92.50
60	120	0.01	0.11	0.23	0.22	112.77
60	60	0.00	0.09	0.13	0.11	150.51
120	480	0.01	0.29	0.18	0.68	99.07
120	240	0.03	0.18	0.26	0.39	127.41
120	120	0.01	0.13	0.26	0.33	160.89

Source: Appendix G.

6.3.2 Effect on Utilization

Figure 6.12 shows the utilization for stations 3, 6, 10, and 13 as a function of MTR, for the four levels of MTBF investigated, 60, 120, 240, and 480 minutes.

For all stations, Figure 6.12 shows that for any fixed MTR the utilization decreases as MTBF increases and that for any fixed MTBF the utilization decreases as MTR increases.

6.3.3 Effect on the Length of the Run Time

Figure 6.13, shows that the time needed to achieve the production requirements increases with MTR, and a big degradation occurs as MTBF increases. These values are in reference to the initial run time without failures which was 588.05 minutes (see Figure 6.5). The production levels were kept as in Example 6.1.

6.3.4 Effect on the Station Buffers

As discussed in section 6.2, when malfunctions occur the dispatching policy preempts the loading of parts towards the failed stations. Furthermore, the loading frequency is tuned to avoid an input faster than the overall throughput.

Thus, as shown in Figure 6.14, these average levels are kept low even for very high MTR or MTBF, and no specific trend exists otherwise.

UTILIZATION

197

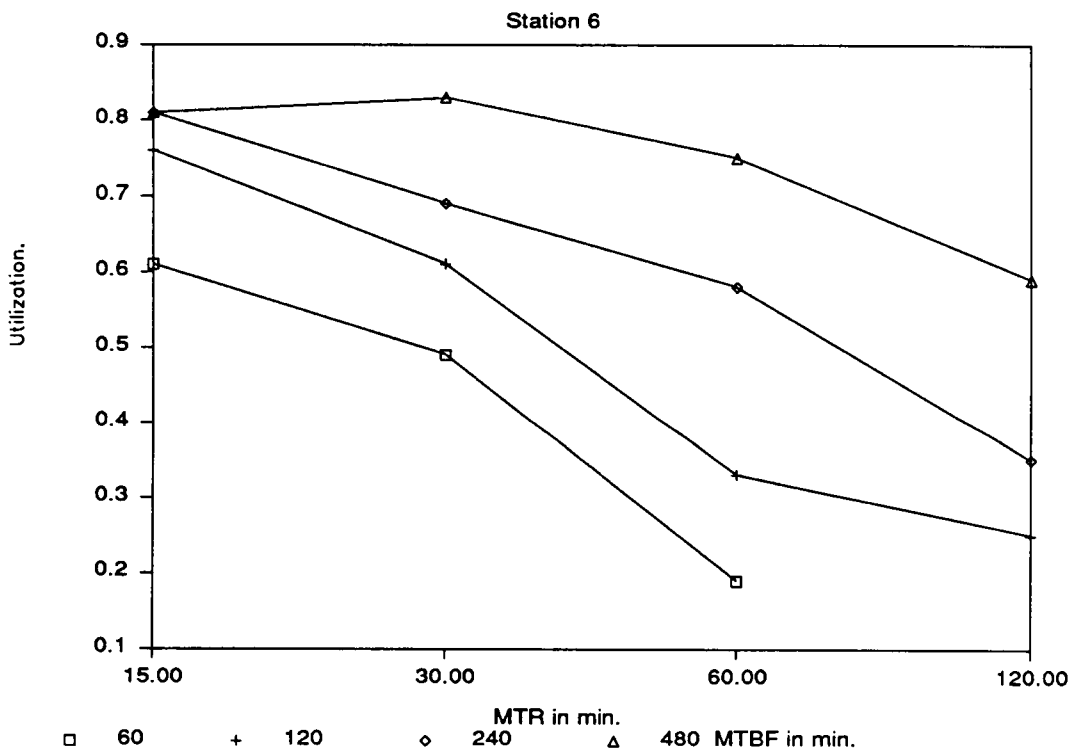
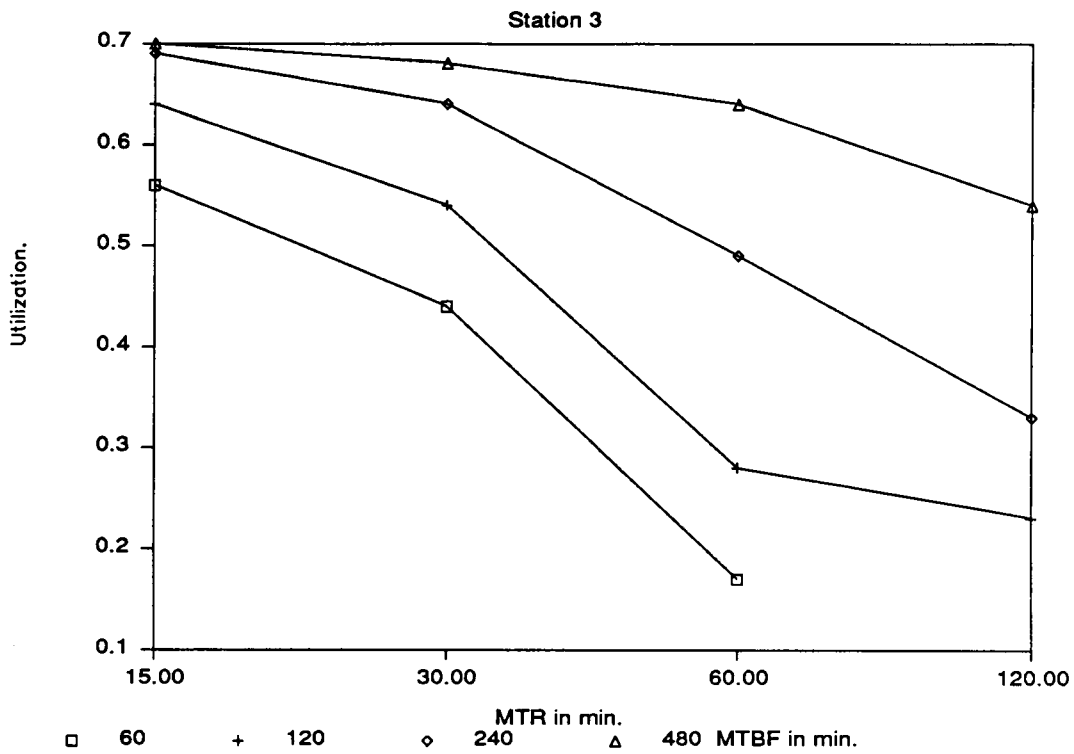


Figure 6.12: Station Utilization Given MTR and MTBF Levels.

UTILIZATION

198

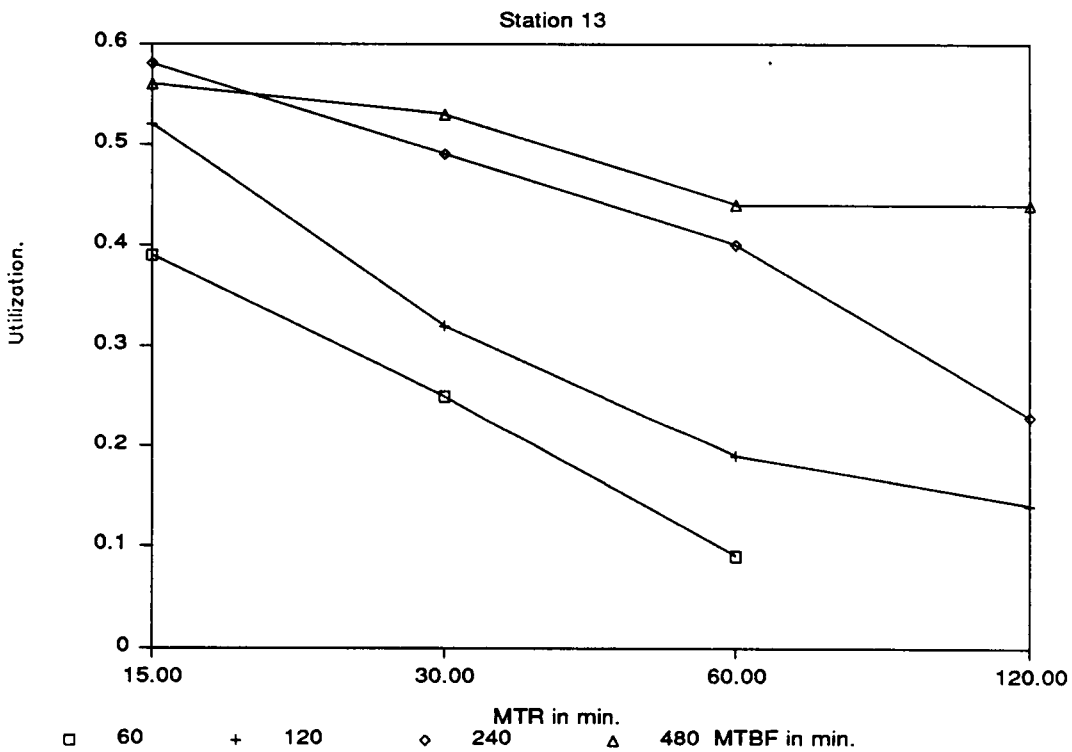
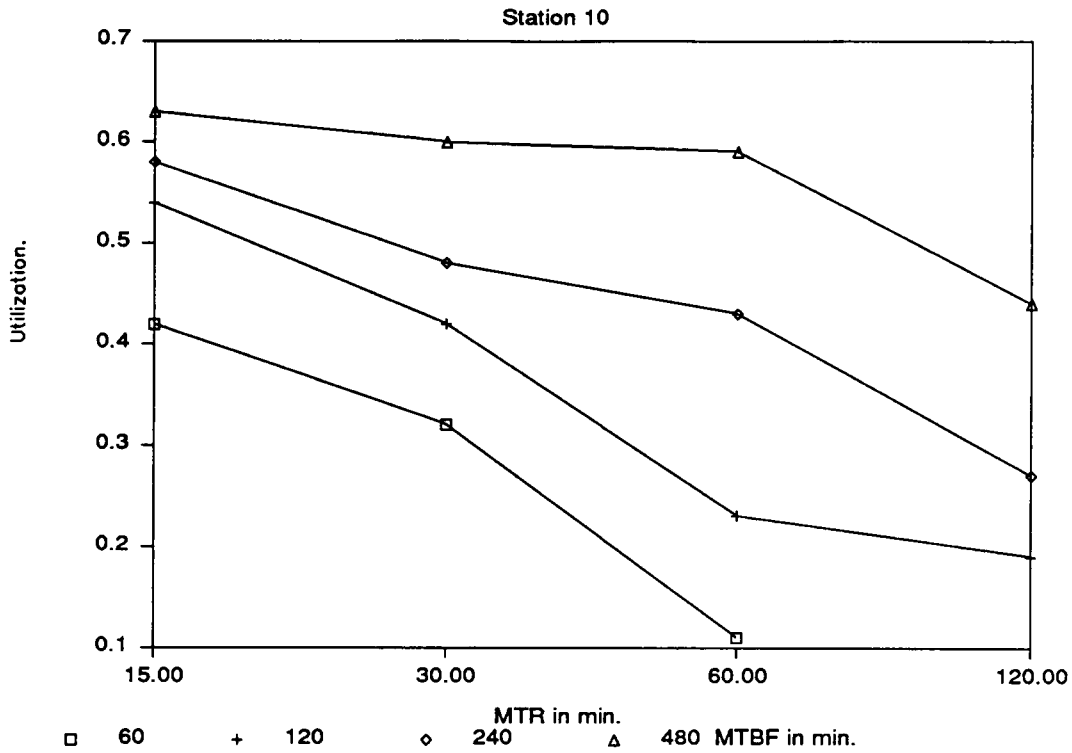


Figure 6.12, continued.

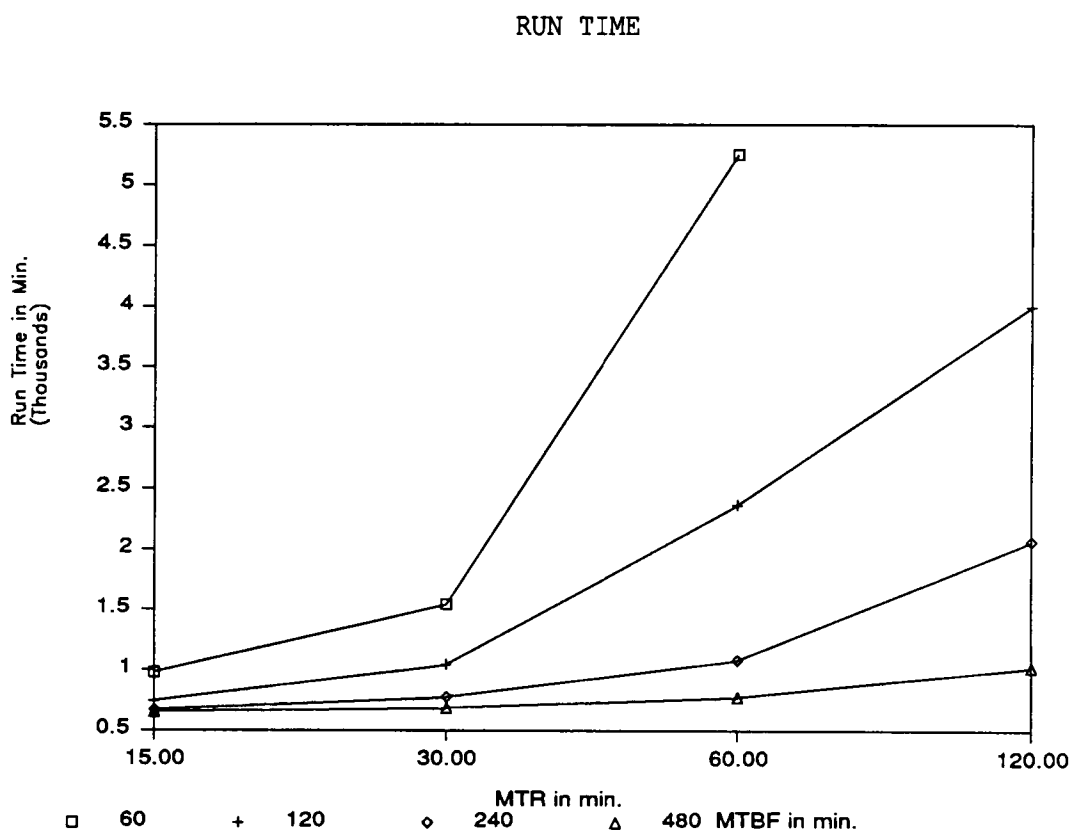


Figure 6.13: Length of the Run Time Given MTR and MTBF Levels.

STATION BUFFERS

200

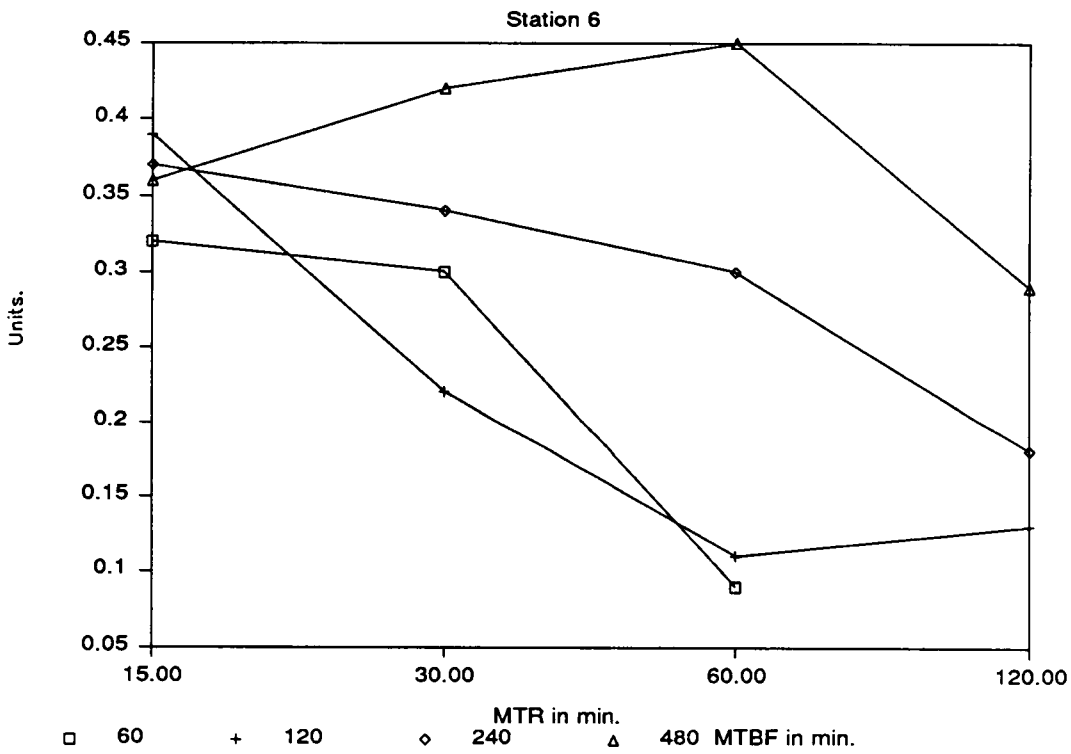
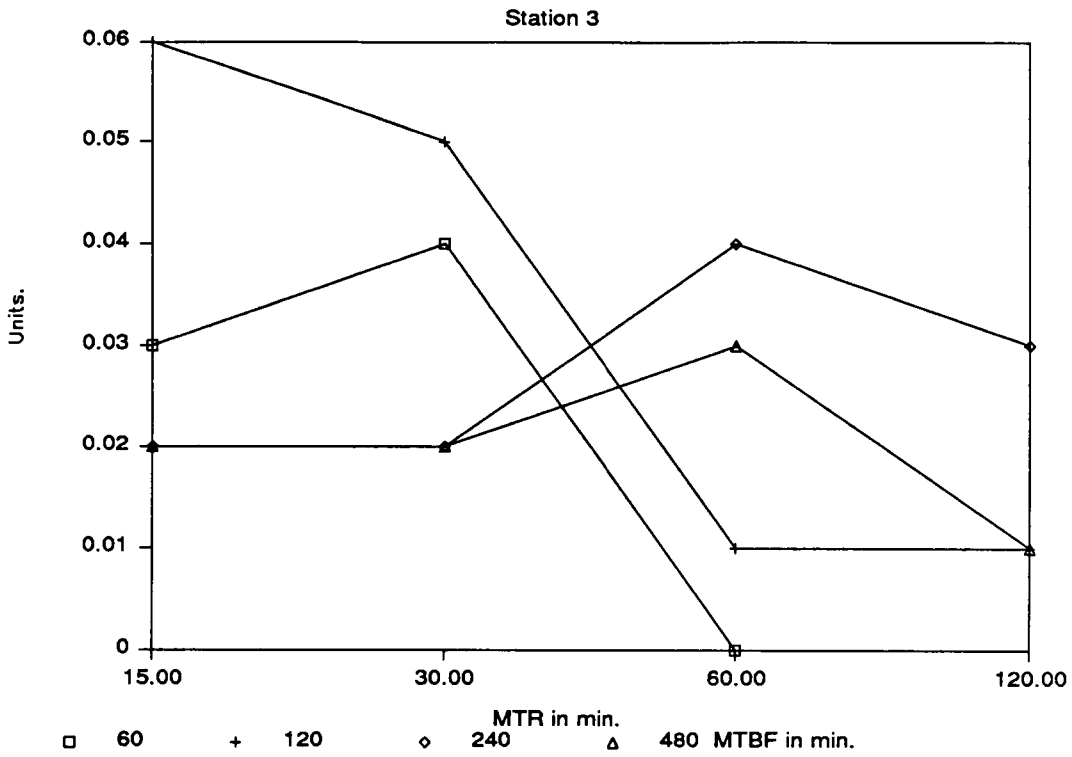


Figure 6.14: Buffer Size Given MTR and MTBF Levels.

STATION BUFFERS

201

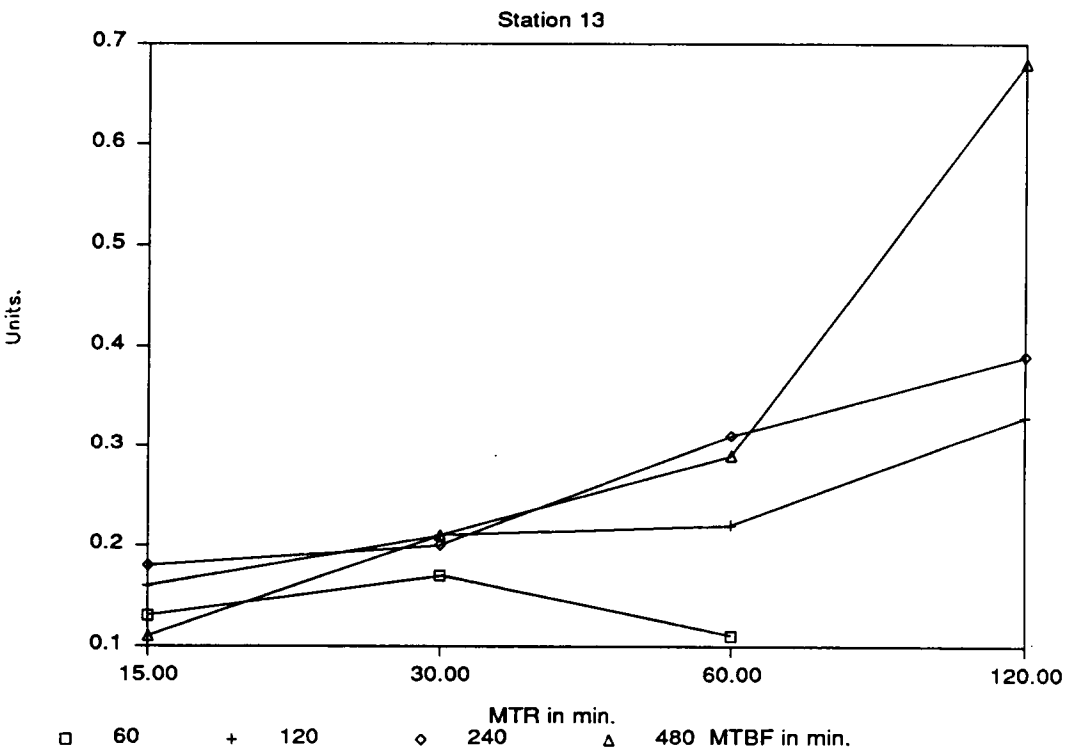
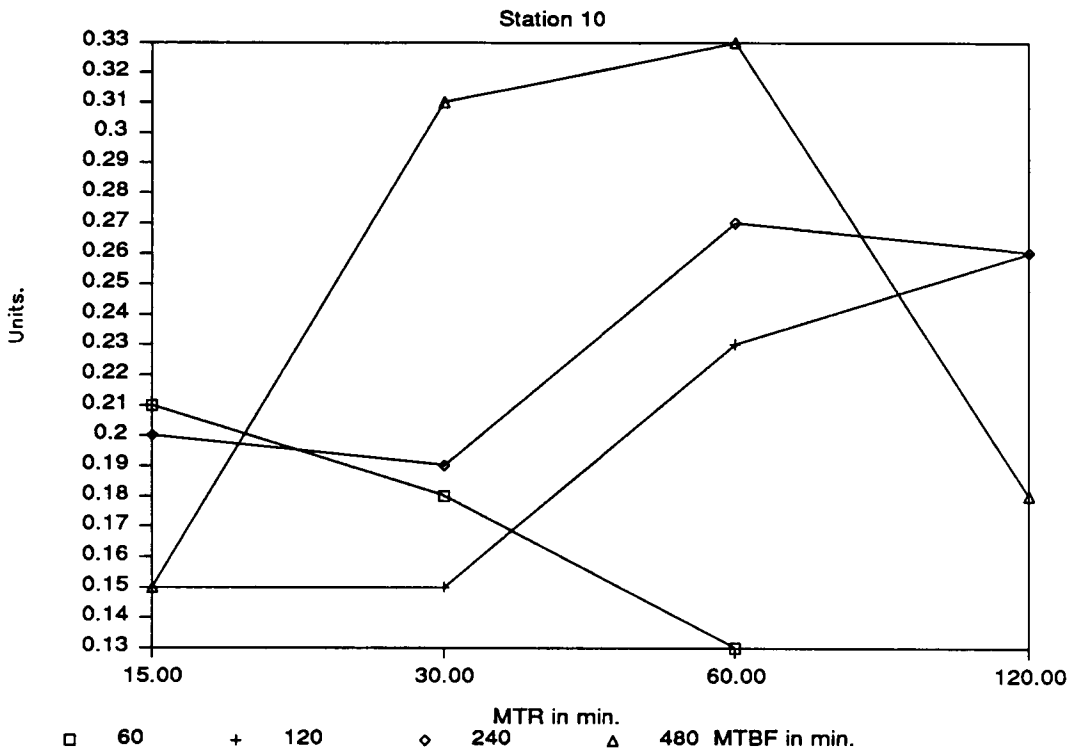


Figure 6.14, continued.

6.3.5 Effect on the Time in System

Figure 6.15 shows that the time the parts spend in the cell follows a similar trend to that of the run time. That is, it worsens as MTR, MTBF, or both increase. Parts in the cell have to wait longer in the buffers or at the stations as the failures are more frequent or the time to repair the equipment is larger.

6.4 Theoretical Justification and Approximate Formulas

Consider the LP in (5.14) with $DF = 1$,

$$\begin{aligned} & \text{Min } \sum_j c(j,t) \cdot u(j,t) && (j=1,\dots,N) \\ \text{st: } & \sum_j P(i,j) \cdot u(j,t) \leq a(i,t) && (i=1,\dots,M) \\ & u(j,t) \geq 0, \text{ for all } j. \end{aligned} \quad (6.1)$$

$$a(i,t) = \begin{cases} 1 & \text{if station } i \text{ is working at time } t, \\ 0 & \text{otherwise} \end{cases}$$

Let $0 < t_1 < t_2 < \dots < t_{k-1} < t_k < \dots < t_T = \tau$,

be a partition over time t , $t \in [0, \tau]$; $\delta t_k = t_k - t_{k-1}$;

$\sigma_k \in [t_{k-1}, t_k]$.

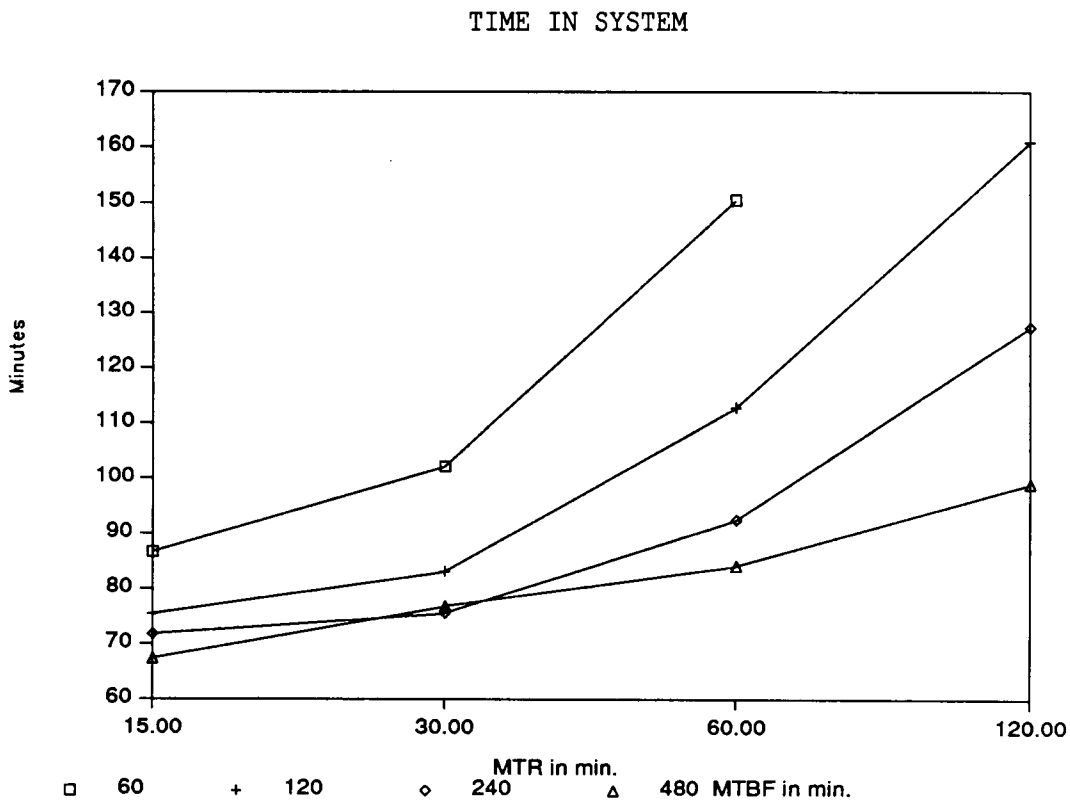


Figure 6.15: Time in the System Given MTR and MTBF Levels.

Then, $\sum_j P(i,j) u(j, \sigma_k) \delta t_k \leq a(i, \sigma_k) \delta t_k \quad (i=1, \dots, M)$
 and,

$$\sum_k \sum_j P(i,j) u(j, \sigma_k) \delta t_k \leq \sum_k a(i, \sigma_k) \delta t_k,$$

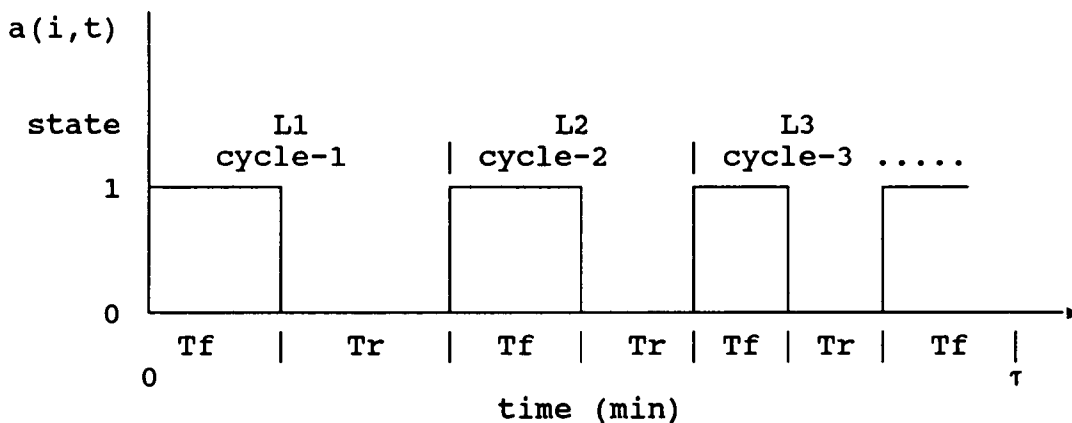
in the limit as δ_k goes to 0,

$$\sum_j P(i,j) U(j, \tau) \leq \int_0^\tau a(i, s) ds = \alpha(i, \tau), \quad (6.2)$$

where $U(j, \tau)$ is the production of part type j up to time τ ,

$$U(j, \tau) = \int_0^\tau u(j, s) ds. \quad (6.3)$$

The function $\alpha(i, \tau)$ is the total time station i has been operating up to a time τ . $\alpha(i, \tau)$ is a random variable for a given τ , that changes as shown in Figure 6.16.



Tf: Time Between Failures.

Tr: Time to Repair.

L_k : Length of cycle k , $k=1, \dots, \infty$.

Figure 6.16: Change of Station State in Time

Note that the conditional probability,

$$P [\alpha(i, \tau) < \alpha \mid \{NC(\tau) = n\}] = \frac{P [\{\sum_{0}^n T_f < \alpha\} \cap \{NC(\tau) = n\}]}{P [NC(\tau) = n]} \quad (6.4)$$

with $NC(\tau)$ being the number of cycles (renewals) in $[0, \tau]$. Since each $L_k = T_f + T_r$, the sum of n cycles is distributed according to the density function,

$$\phi(1) = [f(t_f) * f(t_r)]^{(n)} \quad (6.5)$$

which is the n -th convolution of the convolution of $f(t_f)$ and $f(t_r)$, with the $f(\cdot)$ being the densities of T_f and T_r .

From basic renewal processes theory it can be shown that

$$P [N(\tau) = n] = P(L_n \leq \tau) - P(L_{n+1} \leq \tau) \quad (6.6)$$

In order to find $E [\alpha(i, \tau)]$ from (6.4), we would need to find the density for $N(\tau)$ from (6.5) and (6.6) and then find the joint density in (6.4) (the events in the numerator are not independent). The problem, however, is that expression (6.5) leads to a complicated inverse Laplace transform which in general cannot be solved analytically, nor the density of the conjunction.

An approximation to $E [\alpha(i, \tau)]$ can be obtained by observing that in a time τ the average length of each cycle is,

$$L_c = m_{tbf} + m_{tr} \quad (6.7)$$

with $m_{tbf} = E[T_f]$, and $m_{tr} = E[T_r]$. Furthermore, we are assuming that T_f and T_r are the same for all stations. Otherwise, values for each station are needed.

The average number of cycles for station i is,

$$N_c = \frac{\tau}{mtbf + mtr}$$

$$\text{and,} \quad E [\alpha(i, \tau)] \approx \frac{mtbf \cdot \tau}{mtbf + mtr} = \frac{\tau}{1 + \frac{mtr}{mtbf}} \quad (6.8)$$

(See Appendix H for an alternative formal derivation of (6.8) when T_f and T_r are distributed exponentially.)

Now, going back to inequality (6.2), and taking expected values, we get:

$$\sum_j P(i, j) E [U(j, \tau_i)] \leq \frac{\tau_i}{1 + \frac{mtr}{mtbf}} \quad (6.9)$$

(Note that we have introduced a subscript in the right-hand side since we are specifically referring to the term corresponding to the i -th station.) For short, let us call the number

$$1 + \frac{mtr}{mtbf} = r$$

The left-hand side of (6.9) is the expected total time allocated to station i over $[0, \tau_i]$.

6.4.1. Bound for the Station Utilization

Assuming that delays in transportation and setup are negligible as compared to machining times, an upper bound for the utilization of a station i is given by,

$$\text{Util}(j, \tau_i) \leq \frac{1}{\tau_i} \cdot \sum_j P(i, j) E [U(j, \tau_i)] \leq 1/r \quad (6.10)$$

For the bottleneck machine,

$$\text{Util}(j, \tau_i) \approx 1/r \quad (6.11)$$

For $\text{mtbf}_0 < \text{mtbf}_1 < \dots < \text{mtbf}_q$, the number $1/r$ defines a family of curves as shown in Figure 6.17.

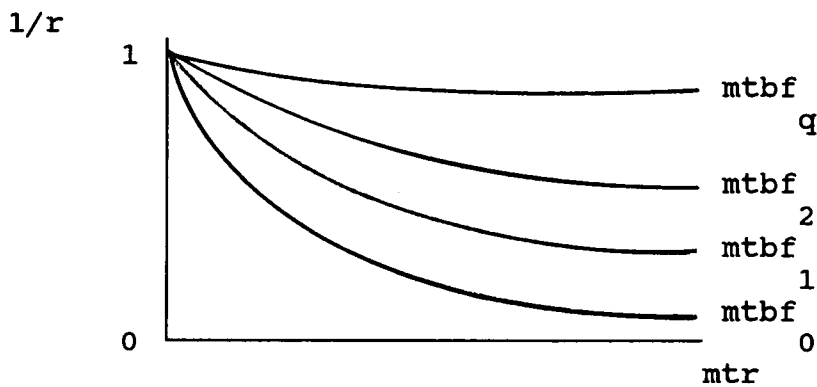


Figure 6.17: Plot of the $(1/r)$ Factor.

Expression 6.11 thus explains the behavior of the utilization shown in Figure 6.12.

6.4.2 Approximation for the Run Time

Let $R(j)$ be the requirement of part j , $j=1,\dots,N$. And, RT_0 be an estimate of the run time in the case when no failures occur. As known,

$$RT_0 = \text{Max}_i \{ \sum_j P(i,j) R(j) \} \quad (6.12)$$

Let RT_1 be the actual time incurred in producing the $R(j)$'s. From (6.9) the following holds:

$$\text{Max}_i \sum_j P(i,j) E[U(j,RT_1)] \leq \frac{RT_1}{r} \quad (6.13)$$

Choose $RT_1 = \text{Min} \{RT\}$ such that $E[U(j,RT_1)] = R(j)$. Then,

$$RT_0 = \text{Max}_i \sum_j P(i,j) R(j) \leq \frac{RT_1}{r} \quad (6.14)$$

$$\text{or,} \quad RT_1 \geq RT_0 \cdot r \quad (6.15)$$

Inequality (6.15) shows that the run time needed to meet the requirements increases at least by a factor,

$$r = (1 + mtr/mtbf).$$

Figure 6.18 depicts how the value of r changes as a function of mtr and $mtbf$.

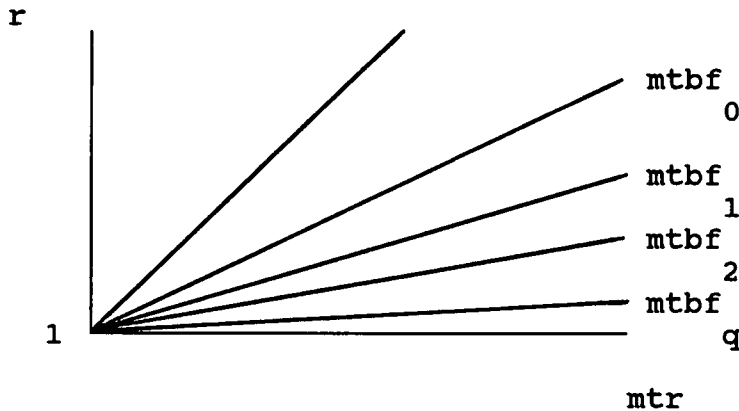


Figure 6.18: Plot of the r Factor.

Where the $mtbf$'s increase downwards in Figure 6.18. As shown earlier in Figure 6.13, the run time rather exhibits an exponential behavior but this may be attributed to sampling error. In any case, the result in formula (6.15) establishes a lower bound and the actual outcome may be larger.

6.4.3 Approximation for the Time in the System

In the case without failures, a lower bound for the time in the system T_{sys} is

$$T_{sys}_0 \geq \frac{\sum_i \sum_j P(i,j) R(j)}{\sum_j R(j)} \quad (6.16)$$

Inequality (6.16) is a lower bound since it ignores, at least, the queuing time in the station buffers. In order to find the lower bound under failures we need to make the following assumption and distinction. First, we assume there is conservation of flow in the cell. When failures occur,

the parts in process are held in the cell; it takes longer to produce them, according to (6.15), but all entered parts exit after completion. Second, we need to distinguish between the nominal allocated time at a station to produce all the parts that visit it, and the actual time the station uses to process the same number of parts after been unavailable due to failures.

The minimum nominal time allocated at a station i is

$$TA_{0,i} = \sum_j P(i,j) R(j) \quad (6.17)$$

The actual time allocated is

$$TA_{1,i} = TA_{0,i} \cdot r, \quad (6.18)$$

since r is the fraction of time the station is available.

Using (6.16) we get,

$$T_{sys_1} = T_{sys_0} \cdot r \quad (6.19)$$

Thus, the time in the system exhibits a behavior similar to that of the run time, as discussed earlier.

CHAPTER 7

CONCLUSIONS

7.1 Summary of the Methodology

In this research we have demonstrated a methodology to develop a knowledge-based system for real-time production control in an automated manufacturing cell. The main aspects of the approach are summarized below:

1. Hierarchical view of CIM functions: overall planning and control in a CIM system is a complex task because too many variables, objectives, and unplanned events must be controlled. The decomposition discussed in Chapter 4 allowed us to define the functions at various layers of a CIM system over different planning horizons. In each layer, the control problem is to satisfy the goals provided by the upper layers while reacting and adapting to unplanned events occurring at the lower levels. This decomposition, as well as the functions, of each layer must be clearly defined in order to identify incoming and outgoing information flows and to limit the scope of control. To the extent that this scope is narrow, knowledge-based systems are feasible to develop.
2. The use of knowledge-based systems: in seeking the automation of decision functions we find that many decisions are knowledge-intensive and ill-structured.

Hence, they are difficult to computerize. In view of this problem we arrived to knowledge-based systems. What separates these systems from other information (or decision support systems) is their ability to incorporate knowledge in a declarative and modular manner. These two characteristics make knowledge-based systems suitable (at least, in the prototyping stage) for incremental development and debugging.

3. The use of model-based knowledge: in conventional expert systems the emphasis is on capturing informal knowledge from human experts (e.g., as "rules of thumb") and encoding it into a knowledge base under a given representation, as reviewed in Chapter 2. For several decades a great amount of theory for analysis of manufacturing systems has emerged from Operations Research (OR) and Simulation. Typical of this developments are the many scheduling models discussed in Chapter 4. Despite of the limitations of these models, namely, over-simplification, planning-oriented instead of real-time-oriented purpose, and slow computing response, they provide valuable sources of knowledge. The model-based approach was used because human experts may be successful in solving static problems but lack the information processing capability to evaluate the result of their actions over the long-run. On the other hand, models are tools to reduce the variety in complex system, but for this very reason they

omit details that can be captured only during the operation of the real system.

7.2 Summary of Contributions

Many modern manufacturing systems have reached a level of computer integration that allow passing messages from to shop floor to a central controller. Operators respond to these messages and send commands in some system-specific syntax. Or, the messages and commands are sent directly from the control program to the shop floor controllers.

In Chapter 5 we described the event-driven inference. Under this type of inference control events are processed as messages which, in conjunction with existing facts in a database, allows the CM to trigger a set of actions. These actions can be messages or commands sent to the shop floor.

The system built under this approach satisfies the requisites of modularity and declarativeness used in knowledge-based systems. Many tasks, however, require procedural knowledge. For this purpose, the rule syntax allows the insertion of function calls.

In addition to writing the set of rules which process the messages and facts, we have provided a way to test the performance of the rule base by linking the system to a simulator. We believe this feature is necessary because the decisions made at different time steps will have an

effect over performance indicators. Thus, we need to test both (a) the correctness of the rule base and (b) its effectiveness. Correctness is the deduction of proper actions, which means verification. Effectiveness is to determine whether the overall performance is satisfactory over a long period.

Since most knowledge-based systems serve their purpose in static problems, the validation process is just a matter of verifying the correct answers. Time-varying systems require, in addition, the analysis of effectiveness.

As mentioned in Chapter 1, the basic problem in real-time production control is how to satisfy the requirements given by the planning stage. We pointed out, in Chapter 4, that most traditional scheduling methods stop in the planning stage but leave the real-time decisions and control of unplanned events to the human scheduler or shop floor manager.

The approach used in the research was first to determine a theoretical production trajectory by using a control model, and second to design a heuristic so that production met the ideal value. The control model accepts feedback since it considers the available capacity at any given time as a function of the stations operating. The objective was to minimize the deviation with respect to a linear demand up to a predefined boundary point, which we defined as the run time. By choosing the run time as the minimum attainable

(the total time allocated at the bottleneck station), the model achieves low flow- and lead-time for the batch. Given the ideal trajectory, the work of the CM was to follow it, but taking into account part routings, delays in the material handling system, and buffers.

The heuristic to match the trajectories resulted in adaptive production ratios for which a simple dispatching policy was adopted (section 5.5.1). Existing work [175] (section 4.5.2) had addressed constant ratios insensitive to capacity changes and worked only for balanced workloads.

In the experiments of Chapter 6, we simulated the CM under different distribution parameters and found that:

1. The mean deviation between predicted and actual production was small (order of 1 unit, at worst). In most cases, this indicator was not significantly ($\alpha = 5\%$) affected by the frequency of failures or length of the repair time (section 6.3.1).
2. Station utilization decreased exponentially with the length of repair times and the failure frequency (sections 6.3.2, 6.4.1).
3. The run time and time in the system increase, at least linearly, with the length of repair times or the frequency of failures (sections 6.3.3, 6.3.5, 6.4.2, 6.4.3).
4. Station buffers are low because the loading of parts using unavailable stations is preempted (section 6.3.4).

7.3 Recommendations for Further Research

1. Implementation issues. The logical continuation of this research is a lower level implementation passing messages from actual or simulated workstations through a LAN. To be consistent with newer trends in manufacturing integration, the message format may be based on the MAP protocol. As discussed in Chapter 5, one limitation of the prototype developed here is that messages come in from and go out to the simulator, and the event catching routine processes the events serially in FIFO order. Actual systems should have the capability to perform multi-tasking and even parallel processing. This poses an important condition the operating system must satisfy.

The use of parallel processing in knowledge-based systems for real-time control may be investigated.

2. Theoretical issues. Regarding the theoretical aspects of the research, the following topics can be further investigated:

(a) We assumed a linear demand rate in model (5.14). The approach can be extended to cases when the demand varies in time. This only requires a change in the function "state" listed in Appendix F.

(b) We assumed that when stations are operating and busy, the incoming parts are put in the station buffer and released in FIFO order. Other heuristic rules could be

tested to see if there is a significant change in performance.

(c) We ignored the effects of pallets and fixtures. Further research can be done to determine rules which rearrange the dispatching sequence in order to minimize pallet and fixture requirements.

LIST OF REFERENCES

REFERENCES

- [1] Adelsberger, H.H. and Neumann, G., 'Goal Oriented Simulation Modeling Using Prolog', AI, Graphics, and Simulation, La Jolla, California, pp.42-47, 1985.
- [2] Adiga, S., 'Artificial Intelligence and its Relevance to Industrial Engineers', 1985 Annual International Industrial Engineering Conference Proceedings, pp.309-314, 1985.
- [3] Adlassnig, K.P., 'Fuzzy Set Theory in Medical Diagnosis', IEEE Transactions on Systems, Man, and Cybernetics, Vol.SMC-16, No.2, pp.260-265, 1986.
- [4] Afentakis, P., Gavish, B. and Karmarkar, U., 'Computationally Efficient Optimal Solutions to the Lot-sizing Problem in Multistage Assembly Systems', Management Science, Vol.30, No.2, pp.222-239, 1984.
- [5] Akella, R., Choong, Y.F. and Gershwin, S.B., 'Performance of Hierarchical Production Scheduling Policy', IEEE Transactions on Components, Hybrids, and Manufacturing Technologies, Vol.CHMT-7, No.3, 1984.
- [6] Al-Zobaidie, A. and Grimson, J.B., 'Expert Systems and Database Systems: How can they serve each other', Expert Systems, Vol.4, No.1, pp.30-37, 1987.
- [7] Albus, J.S. et al., 'A Control System for an Automated Research Facility', Presented at Robots 8 Conference and Exposition, Detroit, Michigan, June 4, 1984.
- [8] Ashby, W.R., 'An Introduction to Cybernetics', Chapman & Hall, London, 1956.
- [9] Baker, T.E. and Collins, D.E., 'The Integration of Planning Scheduling and Control for Automated Manufacturing', Real-time Optimization in Automated Manufacturing Facilities, National Bureau of Standards, Publication No.724, pp.31-58, 1986.
- [10] Barr, A. and Feigenbaum, E.A., 'The Handbook of Artificial Intelligence', Vols. 1-3, Addison-Wesley, Reading, MA, 1982.

- [11] Bellman, R., Esogbue, A.O. and Nabeshima, I., 'Mathematical Aspects of Scheduling and Applications', Pergamon Press, New York, 1982.
- [12] Ben-Arieh, D., 'Knowledge Based Control System for Automated Production and Assembly', unpublished Ph.D. dissertation, Purdue University, 1985.
- [13] Ben-Arieh, D., 'Manufacturing System Application of a Knowledge-Based Simulation', Computers and Industrial Engineering, Whitehouse, G., Eldin, H.K, (eds.), Pergamon Press, NY, Vol.11, No.1-4, pp.459-463, 1986.
- [14] Benton, W.C. and Srivastava, R., 'Product Structure Complexity and Multilevel Lot Sizing Using Alternative Costing Policies', Decision Sciences, Vol.16, pp.357-369, 1985.
- [15] Berry, D.C. and Broadbent, D.E., 'Expert Systems and the Man-Machine Interface', Expert Systems, Vol.3, No.4, pp.228-231, 1986.
- [16] Berry, D.C. and Broadbent, D.E., 'Expert Systems and the Man-Machine Interface. Part Two: The User Interface', Expert Systems, Vol.4, No.1, pp.18-27, 1987.
- [17] Berry, W.L., 'Lot Sizing Procedures for Requirements Planning Systems: A Framework for Analysis', Production and Inventory Management, 2nd. Qtr., pp. 19-34, 1972.
- [18] Billington, P.J., McClain, J.O, and Thomas, L.J., 'Mathematical Programming Approaches to Capacity-Constrained MRP Systems: Review, Formulation and Problem Reduction', Management Science, Vol.29, No. 10, pp.1126-1141, 1983.
- [19] Bitran, G.R. and Matsuo, H., 'The Multi-item Capacitated Lot Size Problem: Error Bounds of Manne's Formulations', Management Science, Vol.32, No.3, pp.350-359, 1986.
- [20] Blackburn, J.D. and Millen R.A., 'Improved Heuristics for Multi-stage Requirements Planning Systems', Management Science, Vol.28, No.1, pp.44-56, 1982.

- [21] Blackburn, J.D. and Millen, R.A., 'Heuristic Lot-sizing Performance in a Rolling-Schedule Environment', Decision Sciences, Vol.11, pp.691-701, 1980.
- [22] Bloch, E., 'Manufacturing Technologies', The Bridge, Fall, pp.10-15, 1985.
- [23] Bloom, H.M. and McLean, C.R., 'Standarization Suggested by the AMRF - A Research Testbed for the Factory of the Future', Presented at First International Symposium on Automated Integrated Manufacturing, San Diego, CA, April 5-6, 1983.
- [24] Bloom, H.M., Furlani, C.N. and Barbera, A.J., 'Emulation as a Design Tool in the Development of Real-Time Control Systems', Presented at the 1984 Winter Simulation Conference, Dallas, TX, November 28-30, 1984.
- [25] Bobrow, D.G., Mittal, S. and Stefik, M.J., 'Expert Systems: Perils and Promise', Communications of the ACM, Vol.29, No.9, pp.880-899, 1986.
- [26] Borgida, A., Greenspan, S. and Mylopoulos, J., 'Knowledge Representation as the Basis for Requirements Specifications', Computer, April, pp.82-90, 1986.
- [27] Borisov, A. and Naglis, L., 'Multi-Criteria Choice of Alternatives in an Expert System for Computer-Aided Design of Industrial Robot Applications', Fuzzy Sets and Systems, Vol.16, pp.93-101, 1985.
- [28] Boudreaux, J.C., 'AMPLE: A Programming Language Environment for Automated Manufacturing', in The Role of Language in Problem Solving-2, J.C. Boudreaux, B.W. Hamill, and R. Jernigan (eds.), North-Holland, pp.359-376, 1986.
- [29] Boudreaux, J.C., 'Problem Solving and the Evolution of Programming Languages', in The Role of Language in Problem Solving-1, R. Jernigan, B.W. Hamill, and D.M. Weintraub (eds.), North-Holland, pp.103-126, 1985.
- [30] Bratko, I., 'Prolog Programming for Artificial Intelligence', Addison-Wesley, Reading, Massachussetts, 1986.

- [31] Bruno, G., Elia, A. and Laface, P., 'A Rule-Based System for Scheduling Production', Computer, July, pp.32-39, 1986.
- [32] Buchanan, B.G., 'Expert Systems: Working Systems and the Research Literature', Expert Systems, Vol.3, No. 1, pp.32-51, 1986.
- [33] Bullers, W.I., Nof, S.Y., and Whinston, A.B., 'Artificial Intelligence in Manufacturing Planning and Control', AIIE Transactions, Vol.12, No.4, pp. 351-363, 1980.
- [34] Burns, N.A., et al., 'The Portable Inference Engine: Fitting significant expertise into small systems', IBM Systems Journal, Vol.25, No.2, pp.236-243, 1986.
- [35] Buzacott, J.A., 'Modelling Automated Manufacturing Systems', Fall Industrial Engineering Conference Proceedings, IIE, 1983.
- [36] Buzacott, J.A. and Shanthikumar, J.G., 'Models for Understanding Flexible Manufacturing Systems', AIIE Transactions, December, pp.339-350, 1980.
- [37] Chand, S., 'A Note on Dynamic Lot Sizing in a Rolling-horizon Environment', Decision Sciences, Vol. 13, 1982.
- [38] Chien, Y. and Liebowitz, J., 'Expert Systems in the SDI Environment', Computer, July, pp.115-122, 1986.
- [39] Chiodini, V., 'A Knowledge Based System for Dynamic Manufacturing Replanning', Real-time Optimization in Automated Manufacturing Facilities, National Bureau of Standards, Publication No.724, September, pp.357-372, 1986.
- [40] Chow, W., MacNair, E.A. and Sauer, C.H., 'Analysis of Manufacturing Systems by the Research Queueing Package', IBM Journal of Research and Development, Vol.29, No.4, pp.330-342, 1985.
- [41] Clayton, B.D., 'ART Programming Tutorial', Inference Corporation, Vol.1, Los Angeles, CA, 1986.
- [42] Cleary, J., Goh, K. and Unger, B., 'Discrete Event Simulation in Prolog', AI, Graphics, and Simulation, Society for Computer Simulation, La Jolla, California, pp.8-13, 1985.

- [43] Clema, J.K., Werling, R. and Chande, A., 'Expert System and Machine Learning', AI, Graphics, and Simulation, Society for Computer Simulation, pp.683-686, 1985.
- [44] Cockett, J.R.B. and Herrera, J.A., 'Prime Rule-based Methodologies Give Inadequate Control', Proc. of the ACM SIGART Int. Symposium of Methodologies for Intelligent Systems, Ras Z.W. and Zemankova M. (Eds.), October, pp.441-449, 1986.
- [45] Davis, D.A., 'Modeling AGV Systems', Proceedings of the 1986 Winter Simulation Conference, pp.568-574, 1986.
- [46] Davis, R., 'Interactive Transfer of Expertise: Acquisition of New Inference Rules', in Artificial Intelligence, Vol.12, North-Holland, pp.121-157, 1979.
- [47] Davis, R. and Lenat, D.B., 'Knowledge-Based Systems in Artificial Intelligence', McGraw-Hill, New York, 1982.
- [48] Davison, J., Schoeder, J.B. and Bell, R., 'Expert Systems Application for Flight Control Maintenance Diagnostics', AI, Graphics, and Simulation, Society for Computer Simulation, pp.1235-1238, 1985.
- [49] De Swaan Arons, H., 'Expert Systems in the Simulation Domain', Mathematics and Computers in Simulation XXV, North-Holland, pp.10-16, 1983.
- [50] Digital Review, 'AI at Work', Vol. 3, No.8, pp.62-69, May, 1986.
- [51] Eden, C.L., 'Rules for Scheduling Semi-automatic Machines with Deterministic Cycle Times', International Journal of Production Research, Vol.13, No.1, pp.41-55, 1975.
- [52] Elmaghraby, A.S. and Jagannathan, V., 'An Expert System for Simulationists', AI, Graphics, and Simulation, Society for Computer Simulation, La Jolla, California, pp.106-109, 1985.
- [53] Engelke, H. et al., 'Integrated Manufacturing Modeling System', IBM Journal of Research and Development, Vol.29, No.4, pp.343-355, 1985.

- [54] Epstein, J.A., 'Intelligently Speaking', Digital Review, May, 1986.
- [55] Faught, W.S., 'Applications of AI in Engineering', Computer, July, pp.17-27, 1986.
- [56] Fellenstein et al., 'A Prototype Manufacturing Knowledge Base in Syllog', IBM Journal of Research and Development, Vol.29, No.4, pp.413-421, 1985.
- [57] Feltner, C.E. and Weiner, S.A., 'Models, Myths and Mysteries in Manufacturing', Industrial Engineering, July, pp.62-76, 1985.
- [58] Fersko-Weiss, H., 'AI Update', Personal Computing, December, pp.99-107, 1986.
- [59] Fink, P.K., Lusth, J.C. and Duran, J.W., 'A General Expert System Design for Diagnostic Problem Solving', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No.5, pp.553-560, 1985.
- [60] Fisher, E.L., 'Expert Systems Can Lay Groundwork For Intelligent CIM Decision Making', Industrial Engineering, March, pp.78-83, 1985.
- [61] Fishman, D., Scott, H. and Bunch, B., 'Integration of Material Buffering Devices in an Automated Factory', Working Paper, National Bureau of Standards, March 17, 1986.
- [62] Florian, M., Tilquin, C. and Tilquin, G., 'An Implicit Enumeration Algorithm for Complex Scheduling Problems', Management Science, Vol.17, No. 12, pp.782-792, 1975.
- [63] Folley, J.D. and Hritz, R.J., 'Embedded AI Expert Systems Troubleshoots Automated Assembly', Industrial Engineering, April, pp.32-36, 1987.
- [64] Ford, D. and Floyd, S., 'An Expert System for Dynamic Scheduling', Proceedings of the ACM SIGART International Symposium for Intelligent Systems, Ras Z.W. and Zemankova, M. (eds.), pp.56-64, 1986.
- [65] Forsyth, R., 'Expertech Xi Plus', Expert Systems, Vol.4, No.1, pp.48-50, 1987.
- [66] Forsyth, R., 'Personal Consultant Plus', Expert Systems, Vol.3, No.4, pp.244-247, 1986.

- [67] Fox, M.S., 'The Intelligent Management System: An Overview', In Processes and Tools for Decision Support, H.G. Sol (ed.), North-Holland, New York, 1983.
- [68] Fox, R.E., 'OPT An Answer for America', Inventories and Production, Vol.2, No.6, November/December, 1982.
- [69] Furlani, C.M., et al., 'The Automated Manufacturing Research Facility of the National Bureau of Standards', Presented at the Summer Computer Simulation Conference, in Vancouver, BC, Canada, July 11-13, 1983.
- [70] Futo, I., 'Combined Discrete/Continuous Modeling and Problem Solving', AI, Graphics, and Simulation, Society for Computer Simulation, La Jolla, California, pp.23-28, 1985.
- [71] Gershwin S.B., Akella, R. and Choong, Y.F., 'Short-term Production Scheduling of an Automated Manufacturing Facility', IBM Journal of Research and Development, Vol.29, No.4, pp.392-400, 1985.
- [72] Gershwin, S.B., 'An Approach to Hierarchical Planning and Scheduling', Real-time Optimization in Automated Manufacturing Facilities, NBS724, Jackson R.H. and Jones (eds.), pp.1-14, 1986.
- [73] Gershwin, S.B. et al., 'A Control Perspective on Recent Trends in Manufacturing Systems', IEEE Control Systems Magazine, Vol.6, No.2, pp.3-15, 1986.
- [74] Grant, T.J., 'Lessons for O.R. from A.I.: A Scheduling Case Study', Journal of the Operational Research Society, Vol.37, No.1, pp.41-57, 1986.
- [75] Hagamen, W.D. and Grady, M., 'The numeric representation of knowledge and logic - Two artificial intelligence applications in medical education', IBM Systems Journal, Vol.25, No.2, pp. 207-235, 1986.
- [76] Haines, C.L., 'An Algorithm for Carrier Routing in a Flexible Material-handling system', IBM Journal of Research and Development, Vol.29, No.4, pp.356-362, 1985.
- [77] Ham, I., Hitomi, K. and Yoshida, T., 'Group Technology Applications to Production Management', Kluwer Academic Publishers Group, Boston, MA, 1985.

- [78] Harmon, P. and King, D., 'Expert Systems. Artificial Intelligence in Business', John Wiley & Sons Inc., 1985.
- [79] Hax, A.C. and Candea, D., 'Production and Inventory Management', Prentice-Hall, Inc., 1984.
- [80] Hax, A.C. and Meal, H.C., 'Hierarchical Integration of Production Planning and Scheduling', in TIMS Studies in the Management Science, Vol.1: Logistics, Geisler, M. (ed.), North-Holland/American Elsevier, 1975.
- [81] Hayes-Roth, F., 'The Knowledge-Based Expert System: A Tutorial', Computer, September, pp.11-28, 1984.
- [82] Hayes-Roth, F., Waterman D.A. and Lenat D.B., 'Building Expert Systems', Addison-Wesley, Reading, Mass., 1983.
- [83] Helman, D.H. and Bahuguna, A., 'Explanation Systems for Computer Simulations', Proc. 1986 Winter Simulation Conference, Wilson, J., Henriksen, J., Roberts, S. (eds.), pp.453-459, 1986.
- [84] Herrod, R.A., 'Industrial Applications of Artificial Intelligence', 1987 IEEE Western Conference on Expert Systems, Proceedings, pp.196-202, 1987.
- [85] Ho, Y.C., Eyler, M.A. and Chien, T.T., 'A New Approach to Determine Parameter Sensitivity of Transfer Lines', Management Science, Vol.29, No.6, pp.700-714, 1983.
- [86] Hodil, E.D., et al., 'Knowledge-based systems in the commercial environment', IBM Systems Journal, Vol.25, No.2, pp.147-158, 1986.
- [87] Hofmann, M., Caviedes, J., Bourne, J., Beale, G. and Brodersen, A., 'Building Expert Systems for Repair Domains', Expert Systems, Vol.3, No.1, pp.4-11, 1986.
- [88] Hwang, S. et al., 'Integration of Humans and Computers in the Operation and Control of Flexible Manufacturing Systems', International Journal of Production Research, Vol.22, No.5, 1984.
- [89] Industrial Networking, Inc., 'User Reference Manual for the INI MAP Machine Interface (MMI)', March, 1987.

- [90] Jackson, H.F. and Jones, A.T., 'Hierarchical Decision Making in the Factory of the Future', Draft Paper, National Bureau of Standards, Gaithersburg, MD, 1986.
- [91] Jarke, M. and Vassiliou, Y., 'Coupling Expert Systems with Database Management Systems', in Artificial Intelligence Applications for Business, W. Reitman (ed.), Ablex Publishing Corporation, Norwood, New Jersey, pp.65-85, 1983.
- [92] Jasany, L.C. and Koelsch, J.R., 'Artificial Intelligence', Production Engineering, November, pp. 62-71, 1986.
- [93] Jelinek, M. and Goldhar, J., 'Economics in the Factory of Future', CIM Review, Winter, pp.21-28, 1986.
- [94] Johnson, L.A. and Montgomery, D.C., 'Operations Research in Production Planning, Scheduling, and Inventory Control', John Wiley & Sons, Inc., New York, 1974.
- [95] Jones, A.T. and McLean, C.R., 'A Proposed Hierarchical Control Model for Automated Manufacturing Systems', Journal of Manufacturing Systems, Vol.5, No.1, pp.15-25, 1986.
- [96] Kearney & Trecker Corporation, 'Manufacturing Systems Application Workbook', Kearney & Trecker Corporation, Milwaukee, WI., 1982.
- [97] Kim, S.H., 'Mathematical Foundations of Manufacturing Science: Theory and Implications', unpublished Ph.D. dissertation, MIT, 1985.
- [98] Kimemia, J. and Gershwin, S.B., 'An Algorithm for the Computer Control of a Flexible Manufacturing System', IEE Transactions, December, 1983.
- [99] Kimura, O. and Terada, H., 'Design and Analysis of Pull System, a Method of Multistage Production Control', International Journal of Production Research, Vol.19, No.3, pp.241-253, 1981.
- [100] Kinney, H.D. and McGinnis, L.F., 'Design and Control of Manufacturing Cells', Industrial Engineering, Vol. 19, No.10, pp.28-38, 1987.

- [101] Kinnucan, P., 'Software Tools Speed Expert Systems Development', High Technology, March, 1985.
- [102] Kinoglu, F., Riley, D. and Donath, M., 'Expert System Model of the Design Process', Autofact 85 Conference Proceedings, Detroit, Michigan, November, pp.7-14,7-27, 1985.
- [103] Kirby, K.E., Sepulveda, J. and Sawhney, R., 'MRP System: Users Manual', Dept. Industrial Engineering, University of Tennessee, Knoxville, January, 1986.
- [104] Kobsa, A., 'Knowledge Representation: A Survey of its Mechanisms ,A Sketch of its Semantics', Cybernetics and Systems, Vol.15, pp.41-89, 1984.
- [105] Korf, R.E., 'Depth-First Iterative-Deepening: An Optimal Admissible Tree Search', Artificial Intelligence, Vol.27, pp.97-109, 1985.
- [106] Kropp, D.H. and Carlson, R.C., 'A Lot-sizing Algorithm for Reducing Nervousness in MRP Systems', Management Science, Vol.30, No.2, pp.240-244, 1984.
- [107] Kumara, S., Soyster, A.L. and Kashyap, R.L., 'An Introduction to Artificial Intelligence', Industrial Engineering, December, 1986.
- [108] Kusiak, A., 'Designing Expert Systems for Scheduling of Automated Manufacturing', Industrial Engineering, Vol.18, No.7, pp.42-46, 1987.
- [109] Lawrence, S.R. and Morton,T.E., 'PATRIARCH: Hierarchical Production Scheduling', Real-time Optimization of Automated Manufacturing Facilities, National Bureau of Standards, Publication No.724, January, pp.87-97, 1986.
- [110] LeClair, S.R., 'Multiexpert Knowledge System Architecture for Manufacturing Decision Analysis', unpublished Ph.D. diss., Arizona State University, 1985.
- [111] LeClair, S.R. and Sullivan, W.G., 'Justification of Advanced Manufacturing Technology Using Expert Systems', Proceedings 1985 Fall Industrial Engineering Conference, Dec.8-11, Chicago, Illinois, pp.362-380, 1985.
- [112] Leary, E.J., 'Decision Support Systems: A Look at Hardware, Software And Planning Procedures', Industrial Engineering, October, 1985.

- [113] Lebow, L. and Blankenship, G.L., 'A Microcomputer Based Expert Controller for Industrial Applications', Draft paper, Systems Research and Electrical Engineering Department, University of Maryland, 1986.
- [114] Liebowitz, J., 'Useful Approach for Evaluating Expert Systems', Expert Systems, Vol.3, No.2, pp.86-96, 1986.
- [115] Lulu, M., 'Characterization of the Just-in-Time Manufacturing and Production System', unpublished Ph. D. diss., University of Alabama, Huntsville, 1985.
- [116] Malmborg, C.J., Agee, M., Simons, G.R. and Choudhry, J.V., 'A Prototype Expert System for Industrial Truck Type Selection', Industrial Engineering, March, pp.58-64, 1987.
- [117] Manuel, T., 'What's Holding Back Expert Systems?', Electronics, August, pp.59-63, 1986.
- [118] Marlin, P.G., 'An MRP/Job Shop Stochastic Simulation Model', Production Management: Methods and Studies, B.Lev (Ed.), Elsevier Science Publishers B.V. (North-Holland), pp.163-172, 1986.
- [119] Masud, A., 'Knowledge-Based Systems and IE', Proceedings 1985 Fall IE Conference, Dec.8-11, Chicago, Illinois, 1985.
- [120] Mayer, R.J., Young, R.E. and Phillips, D., 'Artificial Intelligence Applications in Manufacturing', presented at Autofact 6 Conference, October 1-4, Anaheim, California, 1984.
- [121] McLean, C., Mitchell, M. and Barkmeyer, E., 'A Computer Architecture for Small Batch Manufacturing', IEEE Spectrum, May, pp.59-64, 1983.
- [122] McLean, C.R., and Wenger, C.E., 'The AMRF Material Handling System Architecture', Proceedings of the Fifth Annual Control Engineering Conference, May 6-8, Rosemont, IL, pp.40-49., 1986.
- [123] Melnyk, S.A., Carter, P.L., Dilts, D.M. and Lyth, D. M., 'Shop Floor Control', Dow Jones-Irwin, Homewood, Illinois, 1985.
- [124] Mesarovic, M.D., Macko, D. and Takahara, Y., 'Theory of Hierarchical, Multilevel Systems', Academic Press, New York, 1970.

- [125] Milliken, K.R., et al., 'YES/MVS and the automation of operations for large computer complexes', IBM Systems Journal, Vol.25, No.2, pp.159-180, 1986.
- [126] Minsky, M., 'A Framework for Representing Knowledge', in The Psychology of Computer Vision, Winston, P. (Ed.), McGraw-Hill, New York, pp.211-277, 1975.
- [127] Moily, J.P. and Matthews, J.P., 'Procedures for Determining Relative Frequencies of Production/Order in Multistage Assembly Systems', Decision Sciences, Vol.18, pp.279-291, 1987.
- [128] Murphy, T., 'Artificial Intelligence Topics at IBM', IBM Research Highlights, No.2, 1984.
- [129] National Research Council, 'Computer Integration of Engineering Design and Production, A National Opportunity', National Academy Press, Washington D.C., 1984.
- [130] Nau, D.S and Gray, M., 'Hierarchical Knowledge Clustering: A Way to Represent and Use Problem-Solving Knowledge', Working Paper, Computer Science Department, University of Maryland, November 7, 1986.
- [131] Negoita, C.V., 'Expert Systems and Fuzzy Systems', Benjamin-Cummings Publishing Co. Inc., 1985.
- [132] Newell, A. and Simon, H.A., 'Human Problem Solving', Prentice-Hall, New York, 1972.
- [133] Nilsson, N.J., 'Probabilistic Logic', Artificial Intelligence, Vol.28, pp.71-87, 1986.
- [134] Nisanci, H.I and Sury, R.J., 'Production Analysis by Simulation in a Shoe Manufacturing Factory', International Journal of Production Research, Vol.18, No.1, pp.31-41, 1980.
- [135] O'Keefe, R.M., 'Simulation and Expert Systems - A Taxonomy and Some Examples', Simulation, Vol.46, No. 1, pp.10-16, 1986.
- [136] O'Keefe, R.M., 'Expert Systems and Operational Research - Mutual Benefits', Journal of the Operational Research Society, Vol.36, No.2, pp.125-129, 1985.

- [137] O'Leary, D.E., 'Expert Systems in Production Management', Production Management: Methods and Studies, B.Lev (ed.), Elsevier Science Publishers B. V. (North-Holland), pp.175-185, 1986.
- [138] Orlicky, J.A., 'Material Requirements Planning: The New Way of Life in Production and Inventory Management', McGraw-Hill, New York, 1975.
- [139] Oxman, R. and Gero, J.S., 'Using an Expert System for Design Diagnosis and Design Synthesis', Expert Systems, Vol.4, No.1, pp.4-15, 1987.
- [140] Pau, L.F., 'Survey of Expert Systems for Fault Detection, Test Generation and Maintenance', Expert Systems, Vol.3, No.2, pp.100-111, 1986.
- [141] Paul, R.J. and Doukidis, G.I., 'Further Developments in the Use of Artificial Intelligence Techniques which Formulate Simulation Problems', Journal of the Operational Research Society, Vol.37, No.8, pp.787-810, 1986.
- [142] Pentico, D.W., 'Evaluating Scrap Allowance Decision Rules', Production Management: Methods and Studies, B. Lev (Editor), Elsevier Science Publishers, B.V. (North-Holland), pp.231-249, 1986.
- [143] Phelps, R.I., 'Artificial Intelligence - An Overview of Similarities with O.R.', Journal of the Operational Research Society, Vol.37, No.1, pp.13-20, 1986.
- [144] Pollit, A.S., 'Reducing Complexity by Rejecting the Consultation Model as a Basis for the Design of Expert Systems', Expert Systems, Vol.3, No.4, pp.234-238, 1986.
- [145] Pracht, W.E., 'GISMO: A Visual Problem-Structuring and Knowledge-Organization Tool', IEEE Transactions on Systems, Man, and Cybernetics, Vol.SMC-16, No.2, pp.265-376, 1986.
- [146] Prade, H., 'A Computational Approach to Approximate and Plausible Reasoning with Applications to Expert Systems', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.PAMI-7, No.3, pp.260-279, 1985.

- [147] Rachamadugu, R.V., Raman, N. and Talbot, F.B., 'Real-Time Scheduling of an Automated Manufacturing Center', Real-time Optimization in Automated Manufacturing Facilities, National Bureau of Standards, Publication No.724, pp.293-315, 1986.
- [148] Ramsey, C.L., Reggia J.A., Nau D.S. and Ferrentino, A., 'A Comparative Analysis of Methods for Expert Systems', International Journal of Man-Machine Studies, Vol.24, pp.475-499, 1986.
- [149] Ray, S.R., 'A Knowledge Representation Scheme for Processes in an Automated Manufacturing Environment', Working Paper, Center for Manufacturing Engineering, National Bureau of Standards, 1986.
- [150] Reddy, Y.V., Fox, M.S., Husain, N. and McRoberts, M., 'The Knowledge Based Simulation System', IEEE Software, March, pp.26-37, 1986.
- [151] Rich, E., 'Artificial Intelligence', McGraw-Hill, New York, 1983.
- [152] Richer, M.H., 'An Evaluation of Expert Systems Development Tools', Expert Systems, Vol.3, No.3, pp. 166-183, 1986.
- [153] Robbins, J.H., 'PEPS: The Prototype Expert Priority Scheduler', Autofact 85 Conference Proceedings, Detroit, Michigan, pp.13-10,13-34, 1985.
- [154] Rogers, J.L. and Barthelemy, J.F.M., 'An Expert System for Choosing Options in Programs for Design', in Knowledge-Based Engineering Systems, Research in Progress, IEEE Software, March, pp.51-52, 1986.
- [155] Ruiz-Mier, S., Talavage, J. and Ben-Arieh, D., 'Towards a Knowledge-Based Network Simulation Environment', Proc. 1985 Winter Simulation Conference, Gantz, D. Blais, G. Salomon, S. (eds.), pp.232-237, 1985.
- [156] Russell, R.S. and Taylor, B.W., 'An Evaluation of Scheduling Policies in a Dual Resource Constrained Assembly Shop', IEE Transactions, Vol.17, No.3, pp. 219-232, 1985.
- [157] Russell, R.S. and Taylor, B.W., 'An Evaluation of Sequencing Rules for an Assembly Shop', Decision Sciences, Vol.16, pp.196-212, 1985.

- [158] Sachs, P.A., Paterson, A.M. and Turner, M.H.M., 'Escort-An Expert System for Complex Operations in Real-Time', Expert Systems, Vol.3, No.1, pp.22-28, 1986.
- [159] Sathi, A., Fox, M.S. and Greenberg, M., 'Representation of Activity Knowledge for Project Management', IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No.5, pp.531-552, 1985.
- [160] Schriber, T.J., 'A GPSS/H Model for a Hypothetical Flexible Manufacturing System', Annals of Operations Research, Vol.3, J.C. Baltzer, A.G, Scientific Publishing Company, 1985.
- [161] Scott, H. and Strouse, K., 'Workstation Control in a Computer Integrated Manufacturing System', Working Paper, National Bureau of Standards, Manufacturing Systems Group, Washington, D.C. 20234, 1984.
- [162] Seidmann, A. and Schweitzer, P.J., 'Part Selection Policy for a Flexible Manufacturing Cell Feeding Several Production Lines', IEE Transactions, Vol.16, No.4, pp.355-362, 1984.
- [163] Sepulveda, J.M. and DePorter, E.L., 'IPSS: An Approach Towards Automated Decisions in CIM Systems', Computers and Industrial Engineering, Whitehouse, G., Nazemetz, J.W. (eds.), Pergamon Press, NY, Vol.13, No.1-4, pp.11-14, 1987.
- [164] Sepulveda, J.M. and DePorter, E.L., 'Expert Decision Support System for Production Planning and Control', Proceedings 23rd Annual Meeting of TIMS, Myrtle Beach, SC, October 8-9, 1987.
- [165] Shannon, R.E., Mayer, R. and Adelsberger, H.H., 'Expert Systems and Simulation', Simulation, June, pp.275-284, 1985.
- [166] Shaw, M.J., 'Knowledge-based Scheduling in Flexible Manufacturing Systems', Texas Instruments Technical Journal, pp. 54-61, Winter, 1987.
- [167] Shaw, M.J., 'A Two-Level Planning and Scheduling Approach For Computer Integrated Manufacturing', Real-time Optimization in Automated Manufacturing Facilities, National Bureau of Standards, Publication No.724, September, pp.185-195, 1986.

- [168] Shirai, Y. and Tsujii, J., 'Artificial Intelligence Concepts, Techniques and Applications', John Wiley & Sons, New York, 1982.
- [169] Shpilberg, D., Graham, L.E. and Schatz, H., 'ExpertTAX: An Expert System for Corporate Tax Planning', Expert Systems, Vol.3, No.3, pp.136-149., 1986.
- [170] Shunk, D.L., Sullivan, W.G. and Cahill, J., 'Making the Most of IDEF Modeling - The Triple Diagonal Concept', CIM Review, Fall, pp.12-17, 1986.
- [171] Shunk, D.L., Sullivan, W.G. and Mackulak, G.T., 'Generic Factory Simulator in Support of Computer Integrated Manufacturing Research', Proceedings of the Pittsburgh Simulation Conference, Pittsburgh, PA, April 25, 1985.
- [172] Siegel, S.L., 'Simulation of Scheduling Rules Helps Decision-Making on Various Objectives in Manufacturing Plant', Industrial Engineering, Vol.19, No.10, 1987.
- [173] Simpson, J.A., Hocken, R.J. and Albus, J.S., 'The Automated Manufacturing Research Facility of the National Bureau of Standards', Journal of Manufacturing Systems, Vol.1, No.1.
- [174] Steckle, K.E., 'Design, Planning, Scheduling, and Control of Flexible Manufacturing Systems', Annals of Operations Research, Vol.3, J.C. Baltzer A.G., Scientific Publishing Company, 1985.
- [175] Steckle, K.E., 'Determining Aggregated FMS Production Ratios and Minimum Inventory Requirements', Real-Time Optimization in Automated Manufacturing Facilities, National Bureau of Standards, NBS Publication 724, September, pp. 335-356., 1986.
- [176] Steffen, M.S., 'A Survey of Artificial Intelligence-based Scheduling Systems', 1986 Fall Industrial Engineering Conf., Boston, MA, Dec.7-10, pp.395-405, 1986.
- [177] Sullivan, W.G. and LeClair, S.R., 'Justification of Flexible Manufacturing Systems Using Expert Systems Technology', Autofact 85 Conference Proceedings, November, pp.7-1,7-13, 1985.

- [178] Suri, R., 'An Overview of Evaluative Models for Flexible Manufacturing Systems', Annals of Operations Research, Vol.3, J.C. Baltzer A.G., Scientific Publishing Company, 1985.
- [179] Suri, R. and Dille, J.W., 'A Technique for On-line Sensitivity Analysis of Flexible Manufacturing Systems', Annals of Operations Research, Vol.3, J.C. Baltzer A.G., Scientific Publishing Company, 1985.
- [180] Suri, R. and Whitney, C.K., 'Decision Support Requirements in Flexible Manufacturing', Journal of Manufacturing Systems, Vol.3, No.1, 1984.
- [181] Suri, R., Diehl, G.W. and Dean, R., 'Quick and Easy Manufacturing Systems Analysis Using MANUPLAN', Annual International Industrial Engineering Conference Proceedings, 1986.
- [182] Swyt, D.A., 'U.S. Manufacturing Systems: Factories Past, Present and Future', Working Paper, National Bureau of Standards, Gaithersburg, MD, 1986.
- [183] Symonds, A.J., 'Introduction to IBM's knowledge-systems products', IBM Systems Journal, Vol.25, No.2, pp.134-146, 1986.
- [184] Szabo, S., Juberts, M. and Kilmer, R.D., 'Automated Guided Vehicles at the National Bureau of Standards Manufacturing Research Facility', Unmanned Systems, Summer, pp.38-46, 1984.
- [185] Takenouchi, H. and Iwashita, Y., 'An Integrated Knowledge Representation Scheme for Expert Systems', Expert Systems, Vol.4, No.1, pp.38-43, 1987.
- [186] Talukdar, S.N., Cardozo, E. and Leao, L.V., 'Toast: The Power System Operator's Assistant', Computer, July, pp.53-59, 1986.
- [187] Thomas, L.J. and Weiss, E.N., 'Integrative Approach for Production Scheduling and Capacity Determination', Production Management: Methods and Studies, B.Lev (Editor), Elsevier Science Publishers, B.V. (North-Holland), 1986.
- [188] Turbam, E., 'Developing Expert Systems', Draft paper, Institute of Safety and Systems Management, University of Southern California, June, 1986.

- [189] Turbam, E. and Watkins P.R., 'Integrating Expert Systems and Decision Support Systems', MIS Quarterly, June, pp.121-136, 1986.
- [190] Ulicny, J., et al., 'Knowledge Representation and Artificial Intelligence in the Control of the Hierarchy Production System', Lecture Notes in Control and Information Sciences, Balakrishnan, A.V. and Thoma, M. (eds.), Springer-Verlag, pp.631-637.
- [191] Varney, M.S., Sullivan, W.G. and Cochran, J.K., 'Justification of Flexible Manufacturing Systems with the Analytical Hierarchy Process', 1985 Annual International IE Conference Proceedings, pp.181-190, 1985.
- [192] Vaucher, J.G., 'Views of Modelling: Comparing the Simulations and AI Approaches', AI, Graphics, and Simulation, Society for Computer Simulation, pp.3-7, 1985.
- [193] Voelker, J.A. and Ratica, J.B., 'The genesis of a knowledge-based system', IBM Systems Journal, Vol.25, No.2, pp.181-189, 1986.
- [194] Wagner, H.M. and Whitin, T.M., 'A Dynamic Version of Economic Lot Size Model', Management Science, No.5, pp.89-96, 1958.
- [195] Walker, A., 'Databases, Expert Systems, and PROLOG', in Artificial Intelligence Applications for Business, Reitman W. (ed.), Ablex Publishing Corporation, Norwood, New Jersey, pp.87-109, 1983.
- [196] Waterman, D.A., 'How Expert Systems Differ from Conventional Programs', Expert Systems, Vol.3, No.1, pp.16-19, 1986.
- [197] Waterman, D.A., Paul, J. and Peterson, M., 'Expert Systems for Legal Decision Making', Expert Systems, Vol.3, No.4, pp.212-225, 1986.
- [198] Webster R. and Miner, L., 'Expert Systems, Programming Problem Solving', Technology, January/February, 1982.
- [199] Weitz, R.R., 'Nostradamus: An Expert System for Guiding the Selection and Use of Appropriate Forecasting Techniques', unpublished Ph.D. diss., University of Massachusetts, 1985.

- [200] White, C.C. and Sykes, E.A., 'A User Guided Approach to Conflict Resolution in Rule-Based Expert Systems', IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-16, No.2, pp.276-278, 1986.
- [201] Whitney, C.K. and Suri, R., 'Algorithms for Part and Machine Selection in Flexible Manufacturing Systems', Annals of Operations Research, Vol.3, J.C. Baltzer A.G., Scientific Publishing Company, 1985.
- [202] Williams, J.F., 'Heuristic Techniques for Simultaneous Scheduling of Production and Distribution in Multi-echelon Structures: Theory and Empirical Comparisons', Management Science, Vol.27, No.3, pp.336-352, 1981.
- [203] Winograd, T. and Flores, F., 'Understanding Computers and Cognition, a New Foundation for Design', Addison-Wesley, Reading, Massachusetts, 1987.
- [204] Winston, P.H. and Horn, B.K.P., 'Lisp', Addison-Wesley, Reading, Massachusetts, 1984.
- [205] Wright, P. and Bourne, D., 'Manufacturing Intelligence', Addison-Wesley Pub. Company, Inc., New York, 1988.
- [206] Wysk, R.A., Szu-Yung, W. and Neng-Shu, Y., 'A Multi-Pass Expert Control System (MPECS) for Flexible Manufacturing Systems', Real-time Optimization of Automated Manufacturing Facilities, Nat. Bureau of Standards, Pub. No. 724, pp.251-278, 1986.
- [207] Yano, C.A. and Carlson, R.C., 'An Analysis of Scheduling Policies in Multiechelon Production Systems', IEEE Transactions, Vol.17, No.4, pp.370-377, 1985.
- [208] Yazdani, M., 'Intelligent Tutoring Systems: An Overview', Expert Systems, Vol.3, No.3, pp.154-163, 1986.
- [209] Young, R.E., 'Software Control Strategies for Use in Implementing Flexible Manufacturing Systems', Industrial Engineering, November, 1981.
- [210] Young, R.E. and Mayer, R., 'The Information Dilemma: To Conceptualize Manufacturing As Information Process', Industrial Engineering, September, pp.28-33, 1984.

- [211] Young, R.E. and Rossi, M., 'Knowledge-Based Control of a FMS', Proceedings 1985 Fall IE Conference, Dec. 8-11, Chicago, Illinois, 1985.

APPENDIXES

APPENDIX A

GLOSSARY

A-star: Heuristic search algorithm to find optimal or near-optimal paths in graphs.

Antecedent: The left-hand side of a production rule.

Backward-chaining: A technique used in rule-based systems to control the inference. It starts by selecting a goal hypothesis and proceeds to prove that the conditions leading to that goal are met. See also forward-chaining.

CM: Cell Manager, the system designed in the research.

Consequent: The right-hand side of a production rule containing the conclusions.

Cut: A directive used in the PROLOG language to control the backtracking.

Demon: Procedural element in frame-based systems to perform a sequence of steps whenever its action is requested.

DF: Delay factor, a parameter which adjusts processing times to account for delays in the material handling system.

Directives: Meta-logical element used in a production rule to perform a procedural task.

Emulation: Software design technique which allows a program to receive data from and send data to a device which simulates an actual system.

Event-driven Inference: A control method used in a rule-based system to assert facts and convey state changes when a set of events has occurred.

Facet: An access key denoting a characteristic in the attribute of an object. See also Frame, Slot.

Fail: A control element in the PROLOG language which causes a rule to fail whenever this directive is encountered.

Filter: A LISP function returning a true or nil flag which is used to test if a condition holds.

Firing: The process of asserting all facts whenever the antecedents of a production rule are true.

Flexible Cell: Automated manufacturing system composed of machines and material handling equipment able to manufacture a variety of parts in a family of similar features. It can be reconfigure through software control.

Forward-chaining: A control method used in rule-based systems which starts from an initial set of facts and attempts to derive all new facts. See also backward-chaining.

Frames: A knowledge representation method which describes a prototype object or situation by attributes and properties. Also, the data structure to implement the representation.

Fuzzy Set: Set whose boundary is ill-defined by nature. The degree of association of its elements is measured by a membership function rather than a dichotomic relation.

Heterarchical Control: Software design approach that allows different components of a system to directly exchange messages and assume control in a non-hierarchical manner. The concept can be extended to system analysis.

Hierarchical Planning: The decomposition of long-term, company-wide production plans into shorter-term, more detailed plans at the operational levels.

Inference Control: Strategy to direct the search for conclusions in a knowledge-based system.

Instantiation: The binding of values to the variables of a predicate.

LU: Logical unit.

Logical Unit: An identification code for computer control purpose for stations, stop points, loading/unloading docks, buffers, in an automated manufacturing facility.

Message: A command built with the set of primitives in a computer language grammar.

Meta-modeling: Modeling of an analyst's reasoning when building a mathematical or simulation model of a system.

Model-based Knowledge: The knowledge derived from the study of a system by using a model of it.

Object: A real or abstract entity which is necessary to describe.

Part Mix: The quantity of each part type in production batch.

Path: The list of nodes to visit before reaching a goal.

Pattern-directed Search: The search of facts deduced by instantiation of variables in a predicate-like knowledge representation method.

Pooling: Physical or logical grouping of machines or material handling devices with identical processing or handling capabilities.

Production Rule: Rule, "if-then" rule. Knowledge representation technique with roots in predicate calculus.

Production Ratio: Relative proportion of parts in a mix to be produced over a period of time.

Production Rate: Number of parts produced by unit of time.

Rule Weight: A numeric weight assigned to a rule to indicate the relative ordering in which the rules of the base should be attempted.

Sequence: List of part numbers to be dispatched at any time.

Slot: An access key in a frame denoting an object's attribute.

Variety: System variety, the number of states of a system.

Virtual Cell: A logical, computer re-configurable arrangement or layout of workstations.

Working Memory: Temporary space allocated in the computer memory to store intermediate facts deduced when traversing a knowledge base.

APPENDIX B

A-STAR ALGORITHM

This appendix describes the usage of a LISP function to implement a general purpose heuristic search by using the A* algorithm. The following assumptions have been made:

1. The state space of the problem is represented by the nodes of a graph (tree or net) and the nodes are labeled with any valid Common LISP¹ symbol: numbers, strings, or characters.
2. The start and finish nodes are input parameters.
3. The cost of the arcs in the graph is a function of the two nodes defining the arc. If all costs are equal and constant, the A* algorithm turns into bread-first search.
4. The heuristic function is an estimate of the cost between a given node and the finish (goal) node. When this function is set to zero the search is equivalent to the "optimal search" algorithm. The path will be optimal if the cumulative cost is monotonic non-decreasing.
5. The graph is generated by an expansion function of each node. This function generates the descendants (children) of any parent node. A node without descendants returns nil. Eventually, descendants that have already

¹ For further explanation about these data types and general LISP programming, see "Common Lisp: The Language" by Guy L. Steele Jr., Scott E. Falhman, Richard P. Gabriel, David A. Moon, and Daniel L. Weinreb, Digital Press, 1984.

been generated in an earlier expansion may arise again. This is not a problem since the algorithm checks for cycles and pointers are set adequately depending on whether the new descendants improve the cumulative cost.

The search is performed by invoking the function:

```
asearch (Start Finish Arc-cost
        H-cost Successors)      (B-1)
```

The LISP code for this function is listed in the file LIBRARY.LSP and appears in Appendix F.

The arguments of 'asearch' are as follows:

1. Start: the starting (state) node.
2. Finish: the goal (state) node.
3. Arc-cost: name of the LISP function that defines the cost of an arc. This function has the form:

```
Arc-cost = Arc-cost(node1 node2
                    &optional arg1..argn)      (B-2)
```

Where, the arguments 'node1' and 'node2' are the two nodes (labels) defining the arc, and 'arg1', ..., 'argn' are optional arguments to be defined by the application.

4. H-cost: name of the heuristic function to estimate the cost from a node to the goal. The function has the form:

H-cost = H-cost(node & optional goal) (B-3)

Where, 'node' is the label of the current node from which we want to estimate cost, and 'goal' is the label of the goal state.

5. Successor: name of the function which generates the children of a node. It has the format:

Successor = Successor(node) (B-4)

Where, 'node' is the node being expanded.

The following example shows the use of 'asearch' to find the shortest path between two nodes on the traffic network generated by the logical units of the FMS studied in the research and shown in Figure 1.1.

Example B-1. The code below implements the function 'asearch.' It returns the path between two nodes a and b, as a list: the path to the goal in explicit form. Or, it returns a nil list if the search fails, in which case the user is prompted to input the path. The latter should not occur at all, but the returned value is checked to prevent an error.

```

(defun search-path (a b)
  "Use of A* search for shortest path."
  (let ((path (asearch a b 'lu-distance
                       'lu-hcost
                       'lu-successors)))      (B-5)

    (cond
      ((null path)
       (format t "I have failed to find a path
                  between~%")
       (format t "~S and ~S~%" a b)
       (format t "Please, enter a path ") (read))
      (t path))))

```

The Arc-cost function (B-6) has been labeled 'lu-distance.' This is a weighted cost equal to the distance if the arc (zone) is free, or equal to two times the distance if the arc is unavailable. In order to evaluate the cost, three structures (frames) are kept: (a) zone-id which associates an identification (label) to the arc, (b) zone-queue which stores the queue of vehicles waiting to enter the zone, and (c) lu-distance which is a table of distances between two adjacent logical units, in feet. These structures are listed in the file FLOOR.LSP in Appendix F. In addition, the functions 'fget,' 'hget,' and 'qget' are utilized. These functions simply retrieve values from frames and they are explained in Appendix D.

```

(defun lu-distance (a b &optional (factor 2))
  "Gets weighted distance between two nodes."
  (let* ((zone (hget 'zone-id a b)))      (B-6)
    (cond ((qget 'zone-queue zone)
           (* factor (hget 'lu-distance a b)))
          (t (hget 'lu-distance a b)))))

```

The heuristic function H-cost has been set to zero so that the assumptions in Nilsson's Theorems hold (Section 2.2.3). With H-cost equal zero, the search always expands the node which minimizes the cumulative distance from the starting node.

```
(defun lu-hcost (n &optional g) 0)                (B-7)
"Heuristic cost set to zero."
```

The Successor function simply reads the property 'next' to node 'n' of the frame 'lu.'

```
(defun lu-successors (n)
  "Gets bag of possible adjacent nodes."          (B-8)
  (fget 'lu 'next n))
```

Since the name of the functions Arc-cost, H-cost, and Successor are variables, the same 'asearch' function can be called with different types of functions within a program and for different purposes.

APPENDIX C

KNOWLEDGE-BASE LISTING

```
;;; knowledge base
```

```
;;; rules when cart is free
```

```
(defrule free-cart0
```

```
if
```

```
  (free-cart ?tnow ?cart at ?pos)
```

```
  (queue cart-assignment ?cart ?queue
    (has-value ?queue))
```

```
  (<- (?task ?dest ?load)
```

```
    (release-from-queue cart-assignment ?cart))
```

```
  (<- ?path (get-path ?pos ?dest))
```

```
then
```

```
  (-> (cart status ?cart moving-assigned
```

```
    task ?cart ?task
```

```
    destination ?cart ?dest
```

```
    path ?cart (rest-of ?path)
```

```
    load ?cart ?load))
```

```
  ($ (move-cart ?tnow ?cart from ?pos to ?dest))
```

```
  (exit))
```

```
;;;
```

```
(defrule free-cart1
```

```
if
```

```
  (free-cart ?tnow ?cart at ?pos)
```

```
  (queue cart-assignment any ?queue
```

```
  (has-value ?queue))
```

```
  (<- (?task ?dest ?load)
```

```
    (release-from-queue cart-assignment any))
```

```
  (<- ?path (get-path ?pos ?dest))
```

```
then
```

```
  (-> (cart status ?cart moving-assigned
```

```
    task ?cart ?task
```

```
    destination ?cart ?dest
```

```
    path ?cart (rest-of ?path)
```

```
    load ?cart ?load))
```

```
  ($ (move-cart ?tnow ?cart from ?pos to ?dest))
```

```
  (exit))
```

```
;;;
```

```

(defrule free-cart2
if
  (free-cart ?tnow ?cart at ?pos)
  (lu unloading is ?dock (~ = ?dock ?pos))
  (cart last-position ?cart ?lastp)
  (zone-id ?lastp ?pos ?zone)
then

  (if
    (queue zone-queue ?zone ?queue (has-value ?queue))
    then
      (-> (cart status ?cart stopped-idle))
      ($ (release-cart ?tnow at ?pos)))

  (if
    (queue zone-queue ?zone ?queue (has-no-value ?queue))
    then
      (-> (cart status ?cart stopped-idle
                task ?cart
                load ?cart _))))

;;;

(defrule free-cart3
if
  (free-cart ?tnow ?cart at ?dock)
  (lu unloading is ?dock)
  (lu staging is ?stage-pos)
then
  (-> (cart task ?cart stage))
  ($ (move-cart ?tnow ?cart from ?dock to ?stage-pos)))

;;;

(defrule release-cart1
if
  (release-cart ?tnow at ?pos)
  (cart position ?cart ?pos)
  (cart status ?cart stopped-idle)
  (lu staging is ?stage-pos)
  (<- ?path (get-path ?pos ?stage-pos))
then
  (-> (cart status ?cart moving-idle
            task ?cart stage
            destination ?cart ?stage-pos
            path ?cart (rest-of ?path)
            load ?cart _))
  ($ (move-cart ?tnow ?cart from ?pos to ?stage-pos))
  (exit))

;;;

```

```

(defrule release-cart2
if
  (release-cart ?tnow at ?pos)
  (cart position ?cart ?pos)
  (cart status ?cart ?st)
then
  (cannot release cart ?cart at ?pos status is ?st)
  (exit))
;;;

(defrule release-cart3
if
  (release-cart ?tnow at ?pos)
then
  (no cart at ?pos))
;;;

(defrule load-part0
if
  (load-part ?tnow)
then
  (response to load-part at ?tnow is (get-part-type)))
;;;

(defrule load-part01
if
  (load-part ?tnow)
  (response to load-part at ?tnow is ?ans
    (~ = ?ans no-more-parts-in-batch))
  (time load-step is ?step)
then
  ($ (load-part (+ ?tnow ?step))))
;;;

(defrule load-part1
if
  (response to load-part at ?tnow is ?type)
  (part in-process is ?type)
  (<- ?part (release-from-queue batch ?type)
    (has-value ?part))
  (part operations ?type ?ops)
  (lu loading is ?dock)
  (<- ?station (get-next-machine ?ops))
then
  (-> (load ?part type ?type
    ?part rem-operations ?ops))
  (part ?part loaded at ?tnow at ?dock assigned
    to station ?station))
;;;

```



```

(defrule load-part2
if
  (part ?part loaded at ?tnow at ?dock assigned
   to station ?station)
then

  (if
   (station status ?station idle)
   then
    (-> (station status ?station waiting-load)))
  (if
   (station status ?station ?st (~= ?st idle))
   then (part ?part sent to buffer at station ?station))

  (-> (load ?part next-station ?station
           ?part time-in ?tnow))
  ($ (cart-request ?tnow at ?dock for load-in ?part))
  (exit))
;;;

(defrule load-part3
if
  (response to load-part at ?tnow is
   no-more-parts-in-batch)
then
  (cannot dispatch at time ?tnow)
  (all parts have been dispatched)
  ;;;

(defrule load-part4
if
  (response to load-part at ?tnow is
   critical-machine-down)
  (part t-buffer ?type ?buffer)
  (part safety-level ?type ?safety (< ?buffer ?safety))
  (machine-group number-up ?group 0)
  (part operations ?type ?ops)
  (operation ?x (member ?x ?ops) machine ?group)
then
  (part ?type uses group ?group)
  (no machine available in group ?group at time ?tnow))
  ;;;

(defrule load-part5
if
  (response to load-part at ?tnow is
   buffers-above-safety-level)
  (time retry-load-step is ?step)
then
  (buffers above safety level at time ?tnow)
  ($ (load-part (+ ?tnow ?step))))
  ;;;

```

```

(defrule load-part6
if
  (response to load-part at ?tnow is
    do-not-know)
then
  (shutdown))
;;;

(defrule unload-part1
if
  (unload-part ?tnow ?part)
  (load ?part type ?ptype)
  (load ?part time-in ?tin)
  (part production ?ptype ?n)
  (part t-production ?ptype ?tp)
  (part requirement ?ptype ?r)
  (<- ?prod (+ ?n 1))
then
  (collect s-production ?ptype ?tnow ?prod)
  (collect s-t-production ?ptype ?tnow ?tp)
  (collect deviation ?ptype (- ?tp ?prod))
  (collect tsys part (- ?tnow ?tin))
  (-> (part production ?ptype ?prod
    %-completed ?ptype (/ (* 100 ?prod) ?r))))
;;;

(defrule unload-part2
if
  (unload-part ?tnow ?part)
  (load ?part ?prop ?value)
then
  (fremove load ?part ?prop ?value))
;;;

(defrule unload-part3
if
  (unload-part ?tnow ?part)
  (load ?part type ?ptype)
  (part requirement ?ptype ?r)
  (part production ?ptype ?n (>= (+ ?n 1) ?r))
then
  (fremove part in-process is ?ptype)
  (check-end-run))
;;;

```

```

(defrule start-process1
if
  (start-process ?tnow at ?station of ?part)
  (load ?part rem-operations ?ops)
  (<- ?current-op (first ?ops))
  (operation ?current-op time ?ptime)
then
  (if (station status ?station ?st (~= ?st down))
    then
      (-> (station begin-current-op    ?station ?tnow
          est-end-current-op ?station (+ ?tnow ?ptime)

          status ?station processing))
      (-> (load ?part rem-operations (rest-of ?ops)))
      ($ (end-process (+ ?tnow ?ptime) at ?station of ?part)))

  (if (station status ?station down)
    (cart position ?cart ?station)
  then
    (station ?station is down part ?part is waiting)
    ($ (free-cart ?tnow ?cart at ?station))
    (-> (station est-rem-time ?station ?ptime))))
;;;

```

```

(defrule end-process1
if
  (end-process ?tnow at ?station of ?part)
  (load ?part rem-operations ?ops)
  (<- ?next (get-next-machine ?ops))
then
  (-> (station begin-current-op    ?station
      est-end-current-op ?station _))
  (logical unit ?next selected at ?tnow for ?part in
    ?station))
;;;

```

```

(defrule end-process2
if
  (logical unit ?next selected at ?tnow for ?part in
    ?station (~= ?station ?next))
then
  (if
    (station status ?next idle)
  then
    (-> (station status ?next waiting-load)))
  (if
    (station status ?next ?st (~= ?st idle))
  then
    (part ?part sent to buffer at station ?next))

```

Rule end-process2, continued,

```

(if
  (lu unloading is ?next)
  then
    (part ?part sent to unloading dock))

(-> (station status ?station waiting-unload))
(-> (load ?part next-station ?next))
($ (cart-request ?tnow at ?station
    for unload-machine _)))

;;;

(defrule end-process3
  if
    (logical unit ?station selected at ?tnow
     for ?part in ?station)
  then
    ($ (start-process ?tnow at ?station of ?part)))
  ;;;

(defrule free-station1
  if
    (free-station ?tnow at ?pos)
    (station status ?pos ?st (~= ?st down))
    (queue buffer ?pos ?queue (has-value ?queue))
    (<- ?units (qcount buffer ?pos))
    (<- ?part (first ?queue))
  then
    (release-from-queue buffer ?pos)
    (collect in-buffer ?pos ?tnow (- ?units 1))
    (-> (station current-load ?pos ?part))
    ($ (start-process ?tnow at ?pos of ?part)))

  ;;;

(defrule free-station2
  if
    (free-station ?tnow at ?pos)
    (station status ?pos ?st (~= ?st down))
    (queue buffer ?pos ?queue (has-no-value ?queue))
  then
    (-> (station status ?pos idle))
    (collect station-util ?pos ?tnow 0))
  ;;;

```

```

(defrule cart-request1
if
  (cart-request ?? at ?pos for ?? ??)
then
  (selected cart is (nearest-cart ?pos)))
;;;

(defrule cart-request2
if
  (cart-request ?tnow at ?pos for ?task ?load)
  (selected cart is ?object (~= ?object none))
  (<- (?cart ?path) ?object)
  (cart status ?cart ?cstat
    (member ?cstat (stopped-idle)))
  (cart position ?cart ?cpos)
then
  (-> (cart destination ?cart ?pos
        status ?cart moving-assigned
        load ?cart ?load
        task ?cart ?task
        path ?cart (rest-of ?path)))
  ($ (move-cart ?tnow ?cart from ?cpos to ?pos))
  (exit))
;;;

(defrule cart-request3
if
  (cart-request ?tnow at ?pos for ?task ?load)
  (selected cart is ?object (~= ?object none))
  (<- ?cart (first ?object))
then
  (put-in-queue cart-assignment (entity-is ?task ?pos ?load)
   ?cart))
;;;

(defrule cart-request4
if
  (cart-request ?tnow at ?pos for ?task ?load)
  (selected cart is none)
then
  (put-in-queue cart-assignment (entity-is ?task ?pos ?load)
   any))
;;;

```

```

(defrule failure0
  if
    (station-failure ?tnow ?station)
    (machine-group station ?group ?station)
    (machine-group number-up ?group ?n)
  then
    (-> (machine-group number-up ?group (- ?n 1)))
    (-> (station status ?station down
              time-failed ?station ?tnow))
    (collect station-util ?station ?tnow 0)
    (calculate-new-rates))
  ;;;

(defrule failure1
  if
    (station-failure ?tnow ?station)
    (machine-group station ?group ?station)
    (machine-group number-up ?group ?n (<= ?n 1))
  then
    (erase-from-sequence-parts-using ?group))
  ;;;

(defrule failure2
  if
    (station-failure ?tnow ?station)
    (station status ?station processing)
    (station current-load ?station ?part)
    (station est-end-current-op ?station ?tend)
    (<- ?ans (user-question1 ?tnow ?station ?part t))
  then
    (process of part ?part at station ?station suspended at
     time ?tnow)
    (answer to resume after repair is ?ans)
    (preempt-event end-process ?tend at ?station of ?part)
    (fremove station est-end-current-op ?station ?tend))

  ;;;
(defrule failure3
  if
    (station-failure ?tnow ?station)
    (answer to resume after repair is t)
    (station est-end-current-op ?station ?tend)
  then
    (-> (station est-rem-time ?station (- ?tend ?tnow)))
  ;;;

```

```

(defrule failure4
if
  (process of part ?part at station ?station suspended at
time ?tnow)
  (answer to resume after repair is nil)
  (load ?part ?prop ?value)
then
  (part ?part removed from ?station at time ?tnow)
  (fremove load ?part ?prop ?value))
;;;

```

```

(defrule failure5
if
  (process of part ?? at station ?station suspended at time
??)
  (answer to resume after repair is nil)
then
  (-> (station current-load ?station
        est-rem-time ?station _)))
;;;

```

```

(defrule failure6
if
  (station-failure ?tnow ?station)
  (<- ?repair-time (get-random repair-time))
then
  ($ (station-restart (+ ?tnow ?repair-time) ?station)))
;;;

```

```

(defrule restart1
if (station-restart ?tnow ?lu)
  (machine-group station ?g ?lu)
  (machine-group number-up ?g ?n)
then
  (-> (machine-group number-up ?g (+ ?n 1)))
  (calculate-new-rates))
;;;

```

```

(defrule restart2
if
  (station-restart ?tnow ?station)
  (station time-failed ?station ?tfailed)
  (station time-down ?station ?tdown)
then

```

Rule restart2, continued

```
(if
  (station current-load ?station ?part (= ?part _))
  (station est-rem-time ?station ?trem)
  then
    (-> (station est-end-current-op ?station (+ ?tnow ?trem)
      status ?station processing))
    (collect station-util ?station ?tnow 1)
    ($ (end-process (+ ?tnow ?trem) at ?station of ?part)))

(if (station current-load ?station _)
  then
    (-> (station status ?station idle))
    ($ (free-station ?tnow at ?station)))

(-> (station time-down ?station (+ ?tdown (- ?tnow
  ?tfailed))))
($ (station-failure (+ ?tnow (get-random failure-time)
  ?station)))

;;;

(defrule record-status1
if
  (record-status ?tnow)
  (part in-process is ?ptype)
  (part production ?ptype ?n)
  (part t-production ?ptype ?tp)
then
  (collect s-production ?ptype ?tnow ?n)
  (collect s-t-production ?ptype ?tnow ?tp))
;;;

(defrule record-status2
if
  (record-status ?tnow)
  (time status-step is ?step)
then
  ($ (record-status (+ ?tnow ?step))))
;;;

;;; end
```


APPENDIX D

SHORT USERS GUIDE

This appendix contains information concerning the use of the knowledge representation (KR) method and inference engine employed in the research. It also explains the steps to run the CM from a LISP environment.

The appendix is organized as follows:

1. Sections D.1 describes the filing system and the purpose of each module.
2. Section D.2 discusses the basic data structure to define objects and describes related functions for data retrieval and maintenance.
3. Section D.3 shows the data structure to define rules and describes related functions.
4. Section D.4 describes the interface to the simulator.
5. Section D.5 describes special variables and objects.
6. Section D.6 shows the steps to run the CM.

D.1 Filing System

The shell used in the implementation of the Cell Manager (CM) consists of various modules which contain the programs of the system and the programs, objects, and rules defined by the user. We distinguish the following modules:

1. DATABASE: file that contains user-defined objects as described in Section D.2 below.
2. KNOWLEDGE BASE: file that contains user-defined rules as described in Section D.3 below.
3. LIBRARY: user-defined functions to perform procedural tasks relevant to the application; listed in file LIBRARY.LSP.

The following modules contain the LISP code internally used by the inference engine. Some of the functions may be used for handling objects and rules and will be referred in this appendix. The code for all modules is shown in Appendix F.

4. FRAMES: functions to handle the objects; listed in file FRAMES.LSP.
5. PATTERN: functions to perform symbolic pattern-matching; listed in file PATTERN.LSP.
6. SSIM: functions to perform combined, continuous or discrete event simulation; listed in file SSIM.LSP.
7. UTILITY: small set of functions for input/output; listed in file UTIL.LSP.
8. FF: functions to execute forward-chaining inference; listed in file FF.LSP.

D.2 Objects

An object is any physical or abstract entity, or a class of entities, represented in a frame. A frame is a four-level hierarchical data structure consisting of: a name, slots, facets, and value(s).

(a) Name: is any alphanumeric symbol to identify an object.

(b) Slot: is any alphanumeric symbol to describe an attribute, or a relation associated with the object.

(c) Facet: is any alphanumeric symbol indicating an aspect relative to the slot.

(d) Value: is any data element associated with a facet of a slot--either an alphanumeric symbol or a list of symbols.

In the definition below a token shown as <symbol> denotes a user-defined symbol. An object is defined by the LISP macro:

```
(defobject <object-name>
  :
  .
  (<slot-i>
    (<facet-i1> <value-i11> ....<value-i1n1>)
    :
    (<facet-ij> <value-ij1> ... <value-inj>)
    :
    (<facet-iq> <value-iq1> ... <value-inq>)))
```

(D-1)

Any number of slots, facets, and values can be defined in an object.

Example D-1. Consider the machining of two steel blocks coded 'part-01' and 'part-02.' Figure D-1 shows the final shape of the parts and their operations. (At this level of detail the coding of the operations has been simplified and we omit information concerning the process plan.) Suppose the production monitoring system must keep track of the parts in process, the sequence of operations, the current operation, the number of finished pieces, and the total requirements. We may encode the data in a frame such as:

```
(defobject part
  (in-process (is part-01 part-02))
  (operations (part-01 (drill-hole chamfer-face))
              (part-02 (drill-hole cut-shoulder-1
                        cut-shoulder-2)))
  (current-operation (part-01 drill-hole)
                    (part-02 cut-shoulder-1))
  (requirement (part-01 10) (part-02 20))
  (production (part-01 3) (part-02 5)))
```

D.2.1 Mapping Properties of Objects onto Facts

In Example D-1 above, we can derive facts such as:

```
(part in-process is part-01)
(part in-process is part-02)
(part operations part-01 (drill-hole chamfer-face))
(part operations part-02 (drill-hole cut-shoulder-1
                        cut-shoulder-2))
```

In general, a fact mapped in this way defines a simple grammar of the form:

Fact:= (<object> <property-slot> <facet> <value>) (D-2)

Facts can also be asserted by constructing phrases on the consequent part of a rule, as described in section D.3.

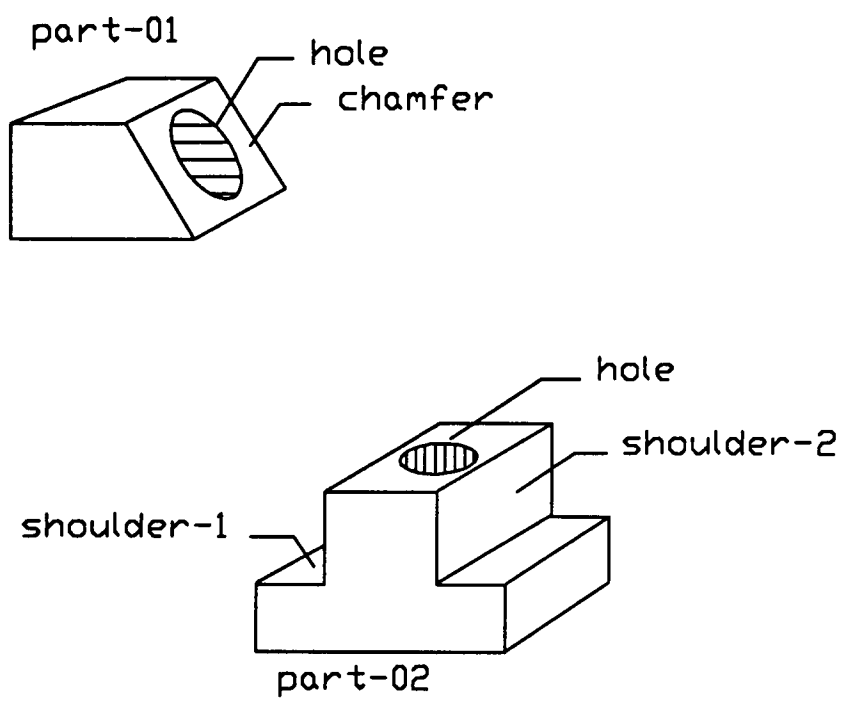


Figure D-1.

There is a number of functions which can be used to update, erase, and perform operations onto the slots and properties of an object, as well as to create and modify objects. These functions are listed in the next section.

D.2.2 Functions Related to Objects

File: FRAMES.LSP

```
-> (frame slot facet value) :
    Replaces a value in a facet of a slot in a frame.

clear-all-frames () :
    Clears all previously defined frames.

defobject (arg1 &rest args) :
    Creates object as defined by structure (D-1).

fget (frame slot facet) :
    Returns a list of values in a facet of a slot in a
    frame.

fget-frame (frame) :
    Returns a frame as defined by structure (D-1).

find-all (predicate) :
    Returns the bag of all instances such that the
    predicate is true, based on all existing frames.
    Needs PATTERN.LSP.

fput (frame slot facet value) :
    Places a value in a facet of a slot in a frame. It
    does nothing if value exists.

fremove (frame slot facet value) :
    Removes a value in a facet of a slot in a frame, if
    value exists.

freplace (frame slot facet value new-value) :
    Substitutes a value with a new-value in a facet of a
    slot in a frame. It does nothing if value does not
    exist.
```

`get-all-frame-facts (&optional
 (a-list (fget 'frame 'frame 'value))) :`
 Returns a list of facts associated with a list of frames. If the input list of frames is not specified, it returns the facts associated with all previously defined frames.

`get-all-queue-facts () :`
 Returns a list of facts associated with all queues. Needs SSIM.LSP (see Section D.4).

`get-object-facts (frame) :`
 Returns a list of facts associated with a single frame.

`get-queue-facts (q) :`
 Returns a list of facts associated with a single queue. Needs SSIM.LSP (see Section D.4).

`hget (frame slot facet) :`
 Returns the first value of `fget` as a symbol instead of a list.

`query (pattern) :`
 Works as `find-all` above but the list is printed.

D.3 Rules

There is a single format for writing a rule:

```
(defrule <rule>
  if
    antecedent-1 antecedent-2... antecedent-n (D-3)
  then
    consequent-1 consequent-2... consequent-m
```

The antecedents of a rule are predicates expressing the conditions under which the rule is true. A rule is said 'to fire' when all its antecedents are true. Upon firing, the consequents correspond to one of the cases below:

1. An intermediate conclusion (fact) to be asserted in the working memory to fire another rule.

2. A message to perform some action.
3. A procedure to be executed.

The rule-based language used in the research is based on the pattern-matching and inference mechanisms proposed by Winston and Horn², Chapters 17 and 18, Part 2. We do not discuss here the basics of this language and the reader is referred to the indicated source. In the research, however, it was necessary to extend the capabilities of these mechanisms in order to perform several tasks not covered in the reference. Modifications include the use of: variables, filters, direct instantiation of a variable, calling procedures by value and by reference, nesting of rules, and search control. Some of these cases are illustrated by means of an example.

Example D-2. When a part type is loaded into the cell to follow a sequence of operations, such part is released from the batch (i.e., a logical queue containing repetitions of the same part type but with different identification codes for individual tracking). The part is located at the loading dock and a procedure gets the next machine to be used for the first operation. An object called 'load' is then created to track the individual part and a message is displayed

²Winston, P.H. and Horn, B.K.P., 'Lisp', Addison-Wesley, Reading, Massachusetts, 1984.

indicating that the part has been loaded and assigned to a station:

```
(defrule load-part1
  if
    (part type loaded at ?tnow is ?type (has-value ?type))
    (<- ?part (release-from-queue batch ?type))
    (part operations ?type ?ops)
    (lu loading is ?lu)
    (<- ?station (get-next-machine ?ops))          (D-4)
  then
    (-> (load ?part type ?type
              ?part operations (rest-of ?ops)))
    (part ?part loaded at ?tnow at ?lu assigned to station
              ?station))
```

Extensions.

1. Variables: are preceded by the question mark '?' and do not require the use of pattern indicators to determine whether the variable requires a new binding or it is restricted to a former instantiation (e.g., ?type, ?ops).

2. Filters: are logical tests to restrict the value of a variable. For instance, (has-value ?type) returns nil if ?type were nil, in which case ?type cannot be bound. Otherwise, the filter would return true with ?type bound to some value.

3. Direct instantiation: variables can be instantiated by calling by value a LISP function. This is done by an antecedent of the type:

```
(<- ?Variable <Expression> &Optional <Filter>),
```

as in:

```
(<- ?station (get-next-machine ?part ?ops)
              (has-value ?station))
```

An <Expression> returning a list can "casted" into a variable list, as in:

```
(<- (?V1 ?V2 .... ?Vn) <Expression>
    &Optional <Filter>)
```

4. Invoking procedures: procedures can be invoked to evaluate an expression within a consequent or to execute a function.

For instance, in:

```
(<-> (load ?part type ?type
        ?part operations (rest-of ?ops)))
```

the function (rest-of ?ops) is called by value to insert the rest of operations, whereas '<->' (defined in D.2.2) is called by reference to update the 'load' frame.

5. Rule factorization: two-level factorization of a rule can be achieved in order to avoid repetition of a set of antecedents for which several separate rules would have to be written. Suppose A, B1, B2,...Bm are groups of antecedents and C1, C2,...,Cm, D are groups of consequents. The m rules:

```
if A B1 then C1 D
if A B2 then C2 D
.
.
if A Bm then Cm D
```

are equivalent to the one rule:

```

if A then
  (if B1 then C1)
  (if B2 then C2)
  .
  .
  (if Bm then Cm)
D

```

6. Control:

(a) When the variable **single-pass** is true, rules are traversed sequentially in the order given by their weights. Otherwise, rules are reset whenever the current rule fires.

(b) The predicate (exit) in the consequents "cuts" the search and no more rules are tried.

D.3.1 Functions Related to Rules

File: FF.LSP

```

== (a b) :
    Same as equal (a b).

```

```

~= (a b) :
    Same as not (equal (a b)).

```

```

deduce () :
    Performs forward-chaining inference with current
    data and knowledge based.

```

```

defact (list-facts) :
    Appends a list of facts to the list *facts*.

```

```

deffun (list-of-function-names) :
    Appends a list of user-defined filter functions to
    the list *oper*.

```

```

defhelp (thing) :
    Attaches a piece of text to a key symbol. The
    argument 'thing' is of the type:
    (<key> <line-1> <line-2> ... <line-p>).
    The arguments <line-1>, ..., <line-p> are strings.
    Text is kept in the variable *explanations*.

```

`defrule (rule &rest args) :`

Defines a rule as shown in structure D-3.

`deftable (table)`

Defines a table whose values can be accessed by the standard 'get' LISP function from within a rule.

The argument 'table' is of the type:

```
(<table-name>
 (<item-1> <value-1>)
 .
 .
 (<item-r> <value-r>)).
```

`defweight (weights) :`

Assigns weights to rules to direct the ordering in which rules are selected for firing. If not defined, rules are accessed in the order they appear in the knowledge base. The argument 'weights' is of the type:

```
((<rule-name-1> <weight-1>)
 .
 .
 (<rule-name-q> <weight-q>)).
```

Weights are kept in variable `*rules-weight*`.

`explain (key) :`

Shows the text associated with the 'key' as defined by 'defhelp.'

`get-facts-subset (event) :`

Asserts in the working memory facts associated with objects associated with the 'event.'

`get-rule-subset () :`

Extracts the rules related to the current event being processed.

`how-fact (fact) :`

Displays the rule or other source the 'fact' came from.

`make-meta (&optional (rules *rules*)) :`

Binds objects and rules to all possible events.
Needs SSIM.LSP (see section D.4).

`reset (&optional (opt 'all)) :`

If opt is all, it clears everything from the working memory. Otherwise, it clears all except the rules.

```

retract (fact) :
    Retracts a fact asserted in the working memory.

run (&optional file) :
    Loads the knowledge base written in 'file.' If
    'file' is defined as "user," rules and facts can be
    asserted interactively.

show-rule (rule) :
    Displays a parsed rule.

show-rule-list (rule) :
    Shows are defined rules.

use-rule (rule) :
    Forward-chains 'rule.'

```

D.4 Simulation Interface

An event is denoted by a predicate of the form:

Event:= (<event-type> <time> &rest args) (D-5)

Where, '<time>' is the time of occurrence of the event, and '&rest args' stands for arguments arg-1, arg-2,..., arg-n relative to the '<event-type>.'

Example D-3. When a failure occurs at a workstation, the number of operational machines in the machine group to which the workstation belongs decreases by one and therefore the capacity of the cell momentarily decreases. This has an effect on the dispatching of all part types which use the machine group. In fact, if there is no redundancy for that workstation (i.e., no pooling) such parts cannot be dispatched at all. In order to determine when, and by how much the

dispatching rate change must change according to the new real-time capacity, the following event can be defined:

(station-failure <time> <station>) (D-6)

The corresponding rule is as follows:

```
(defrule failure-1
  if
    (station-failure ?tnow ?station)
    (machine-group station ?group ?station)
    (machine-group number-up ?group ?n)
  then
    (-> (machine-group number-up ?group (- ?n 1)))
    (-> (station status ?station down
          time-failed ?station ?tnow))
    (calculate-new-rates t))
```

(D-7)

Some time after (within the planning horizon), the workstation will re-start and another rule will calculate again the new production rates. In an actual environment a message would be passed to the CM to make the proper decisions. For simulation purposes, the re-starting time must be generated at the time of failure. This time is:

Re-starting time = Failure Time + Repair Time (D-8)

The repair time may be a function of various factors. A diagnosis (e.g., by using an ES or the repair crew's estimate) would determine the type of failure and the estimated repair time could be fed back to equation (D-8). This process can be represented (or programmed) without

major difficulties, but in absence of the necessary data such a procedure seems somewhat arbitrary to perform. Thus, for experimental analysis the repair time can be simply assumed to be a random variable with a known distribution.

In any case, assuming that the repair time is available either as a deterministic or random number (or may be as a crisp or fuzzy one), the event that accounts for the restart has to be scheduled at the failure time (or after the diagnosis has been made) so that the event manager retrieves it later. This process is similar to that of several discrete event simulation languages. Suppose we define the event as:

```
(station-restart <restart-time> <station>).
```

The rule which schedules the event is as follows:

```
(defrule failure-7
  if
    (station-failure ?tnow ?station)                (D-9)
    (<- ?repair-time (get-random repair-time))
  then
    ($ (station-restart (+ ?tnow ?repair-time) ?station)))
```

Where '(get-random repair-time)' is a procedure that returns the repair time (in this case a random number from a pre-defined distribution). The directive '\$' schedules the event as defined by equation (D-5). This consists of placing the event in the *events* list, sorted in ascending order of the time of occurrence, or in FIFO order in cases of ties.

The inverse of '\$' is 'preempt-event.' This erases an event from *events*. For instance, if ?future-time and ?station have been instantiated in the left-hand side of a rule, the predicate:

(preempt-event station-restart ?future-time ?station),
will preempt the event station-restart.

D.4.1 Relation of Events with Objects and Rules

From Example D-3 it can be seen that a particular event (as defined by (D-5)) may need only a subset of objects (hence, a subset of facts) and also a subset of rules. The inference engine has a built-in ability to detect references to objects and rules which participate in an event. This is only for efficiency purposes: the smaller is the number of facts and rules that have to be searched, the shorter is the time required to arrive to a conclusion or to perform the proper actions.

The predicate '(make-meta)'--listed in D.3.1-- binds events to objects and rules provided that the event antecedent is the first to appear in a rule, as shown in rules (D-7) and (D-9) above.

D.4.2 Functions Related to the Simulation Interface

File: SSIM.LSP

```
$ (event time &rest attr) :  
    Schedules an event defined as a predicate in a rule
```


`clear-all-queues () :`
 Clears all queues defined by 'defqueue' and updated
 by 'put-in-queue' and 'release-from-queue.'

`clear-all-stats () :`
 Clears all statistics defined by 'defstat' and
 updated by 'collect.'

`collect (stat key &optional p1 p2) :`
 Adds an entry to the case 'key' of the statistic
 'stat.'
 Parameters p1 and p2 are as follow:
 If statistic type is 'sample' then p1 is the
 observation.
 If statistic type is 'time-persistent' then p1 is
 the time when the observation is collected and p2 is
 the observation.

`collect-trans (trans) :`
 Keeps a log of events and other transactions:
 queuing and release from a queue.

`defqueue (arg1 &rest args) :`
 Defines a queue by the following structure:

```
(defqueue <queue-name>
  (discipline <discipline>)
  (<key-1>      <initial-queue-1>)
  .
  .
  (<key-n>      <initial-queue-n>))
```

Where <discipline> is fcfs;lcfs.

Note: (1) fcfs=first come first served.

(2) lcfs=last come first served.

(3) other disciplines can be programmed.

(4) if no key is defined it is assumed 'queue.'

`defstat (arg1 &rest args) :`
 Defines a statistic by the following structure:

```
(defstat          <statistic-name>
  (type           <stat-type>)
  (show-values    <show-key>)
  (substract      [<time-down-expression>])
  (<key-1>         <initial-observations-1>)
  .
  .
  (<key-p>         <initial-observations-p>))
```

Where <stat-type> = sample or time-persistent;
 <show-key> = t, if values are shown in reports, or nil otherwise;
 <subtract> = LISP-function returning the time down of resource when collecting utilization.

event-attributes (e) :
 Returns the list of attributes associated with an event 'e.'

event-time (e) :
 Returns the time an event 'e' occurs.

event-type (e) :
 Returns the type of an event 'e.'

get-average (stat key &optional tnow) :
 Returns the mean value of the sample in statistic 'stat' by entry 'key.' If the type of statistic is: 'time-persistent' the average is time-integrated and the optional argument 'tnow' is the upper-limit of the integral. Lower-limit is assumed to be zero.

get-next-event () :
 Returns next event to occur in time and erases it from the list *events*.

get-random (var) :
 Returns a random value from the variable 'var.'

make-seeds () :
 Creates a set of 10 seeds: *seed-1*, *seed-2*, ..., *seed-0*, which can be used in generating random values for different variables.
 Note: In VAX Common LISP the random state is time-dependent. Saving seeds allows replication of simulation experiments with the same set of initial seeds.

preempt-event (event time &rest args):
 Erases from list *events* the event scheduled at some time.

process-event (event) :
 Executes internal logic to process an user-defined event.

put-in-queue (queue entity &optional (key 'queue)) :
 Put an entity in a queue according to the queue discipline.

qcount (queue &optional (key 'queue)) :
 Returns the length of a queue.

qget (queue &optional (key 'queue)) :
 Gets a queue as a list.

read-seeds (f) :
 Retrieves saved seeds from file 'f'.

release-from-queue (queue &optional (key 'queue)) :
 Releases and updates a queue according to the queue discipline.

save-seeds (f) :
 Saves seeds created by 'make-seeds' onto file 'f.'

sched (time type &optional attributes) :
 Schedules an event.

shutdown () :
 Clear all events and stops the simulation.

xsim-all (tbegin &optional (Stepwise nil)) :
 Forward chains all events. If Stepwise is non-nil, the search stops after each event.

xsim-time (tbegin tend &optional (Stepwise nil)) :
 As 'xsim-all' but up to time 'tend.'

D.5 Special Variables and Objects

D.5.1 Special Variables

dist-ops Currently set to '(assertf -> fput \$). These functions are copied on the head of the arguments and allow factorization.

events List of events currently scheduled in the simulator.

explanations Internal representation of explanation list created with defhelp macro.

facts List of facts in working memory.

fcalls List of function calls in working memory.

- *ff-verbose*** If true, it displays conclusions. Set internally to t when running in user mode. Otherwise, it is turned on/off with the trace object (defined below).
- *local-parameters***
 Association list to instantiate parameters in consequents of a rule, setq as:
 '((parm1 value1) (parm2 value2)....(parmk valuek)).
- *no-show*** Currently set to:
 '(fremove collect-stats collect). These functions are not displayed as conclusions.
- *rules-00*** Representation of rules as written in knowledge base.
- *rules-weight***
 Internal representation of the rules' weight created with defweight macro.
- *rules*** Internal representation of rules.
- *s-ops*** Currently set to '(-> \$). These function symbols are not displayed in the consequents preceded by them.
- *show-rule*** If true, it displays in brackets the rule that fired in a conclusion, as [<rule-name>].
- *single-pass***
 If true, it searches the rules sequentially and it does not reset all rules to redo the search.
- *state-variables***
 If true, the simulator will execute the function "state" to update state variables at each time step.

D.5.2 Special Objects

1. To trace events:

- (a) Trace on for events a1, a2, a3,...,an, and off for events b1, b2, b3,...,bm:

```
(defobject trace (on (events a1 a2 a3....an))
                  (off (events b1 b2 b3....bm)))
```

(b) Trace everything except b1, b2, b3,...,bm:

```
(defobject trace (on (events all-events))
  (off (events b1 b2 b3....bm)))
```

2. To process events as defined procedures:

```
(defobject meta-rules
  (procedures (are p1 p2 p3...pm)))
```

D.6 Running the CM Program

This section describes procedures to run the program used in the experiments in Chapter 6. Figure D-2 shows the filing system.

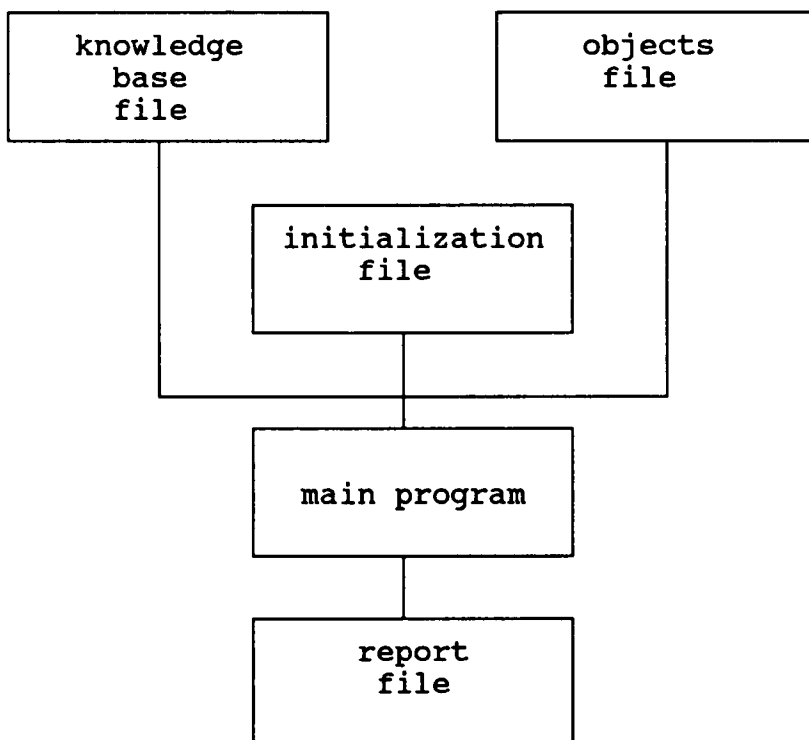


Figure D-2: File System

Two modes to run the CM are available: (a) Mode I: interactive single run, and (b) Mode II: multiple batch run.

D.6.1 Steps in Mode I

1. Build the knowledge base (e.g., KB.LSP, as shown in Appendix C).
2. Build the objects file (e.g., FLOOR.LSP, as listed in Appendix F).
3. Build the initialization file to declare initial events and setup program. See INIT2.LSP in Appendix F.
4. From the LISP environment, call

```
(load "cm")
```

The CM.LSP file is listed in Appendix F.

5. Within CM now, call

```
(load "init2")
```

The file INIT2.LSP is listed in Appendix F.

6. Run the program by using one the options below:

```
(xsim-all 0) : runs until batch is completed or no
               event remains.
```

```
(xsim-time <t0> <t1>) : runs from initial time t0 to
                       end time t1 or until no event
                       remains.
```

7. Save the report file by calling,

```
(report <file-name-string> <report-title-string>)
```

The 'report' LISP function is specific to the problem discussed in Chapter 6, and it is based on the statistics

defined by the macro 'defstat' (section D.4.2) in file FLOOR.LSP. The report function (Appendix F) provides:

(a) production levels, (b) theoretical production levels, (c) deviations between (a) and (b), (d) station utilization, (e) station buffers, and (f) time in the system.

D.6.2 Steps in Mode II

1. Build the knowledge base and name it KB.LSP, as shown in Appendix C.
2. Build the objects file and name it FLOOR.LSP, as listed in Appendix F).
3. Build the initialization file to declare initial events. See INIT.LSP in Appendix F.
4. From the LISP environment, call

```
(load "main")
```

The MAIN.LSP file is listed in Appendix F.

5. Within CM now, run the program by calling,

```
(main <arg1> <arg2> ..... <arg11>)
```

The LISP function main is implemented as follows:

```
main(&optional <run-title>
      <type>
      <run-time>
      <number-seeds-to-try>
      <number-runs-in-each-seed>
      <rtl>
      <rth>
      <mtbf>
      <seed-set-1>
      <seed-set-2>
      <report-file-name>)
```

The meaning of the arguments is as follows:

run-title:
string id. of run (default is blank).

type:
1 = run at least the total time of the bottleneck machine.
2 = run a specified time.

run-time:
if type = 1 enter 0, value is not used.
if type = 2 enter run time in minutes.

number-seeds-to-try:
run is repeated this number of times. Seeds for repair time are chosen from seed-set-1 and those for failure times are chosen from seed-set-2 (default is 1).

number-runs-in-each-seed:
within each seed run this number of replications (default 1).

rtl:
minimum value of uniform distribution for repair time (default 10).

rth:
maximum value of uniform distribution for repair time (default 20).

mtbf:
mean time between failures in exponential distribution (default 240).

seed-set-1:
the default list is (*seed-0**seed-4*).

seed-set-2:
the default list is (*seed-5**seed-9*).

report-file-name:
name of file onto which results are written (default "runs.dat;1").

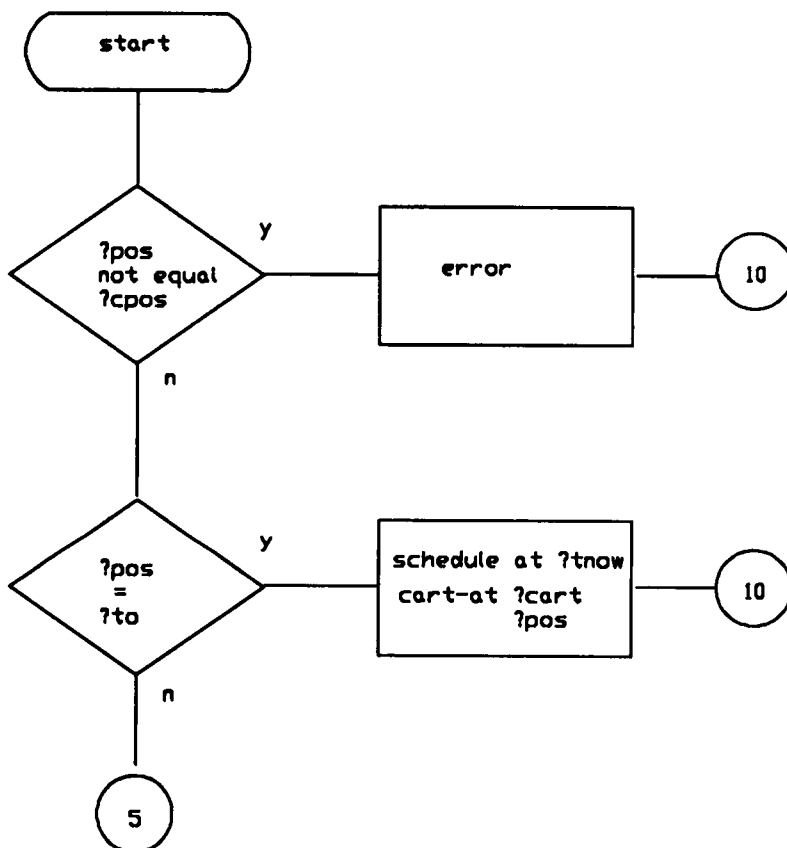
FLOW-CHART FOR AGV SIMULATION

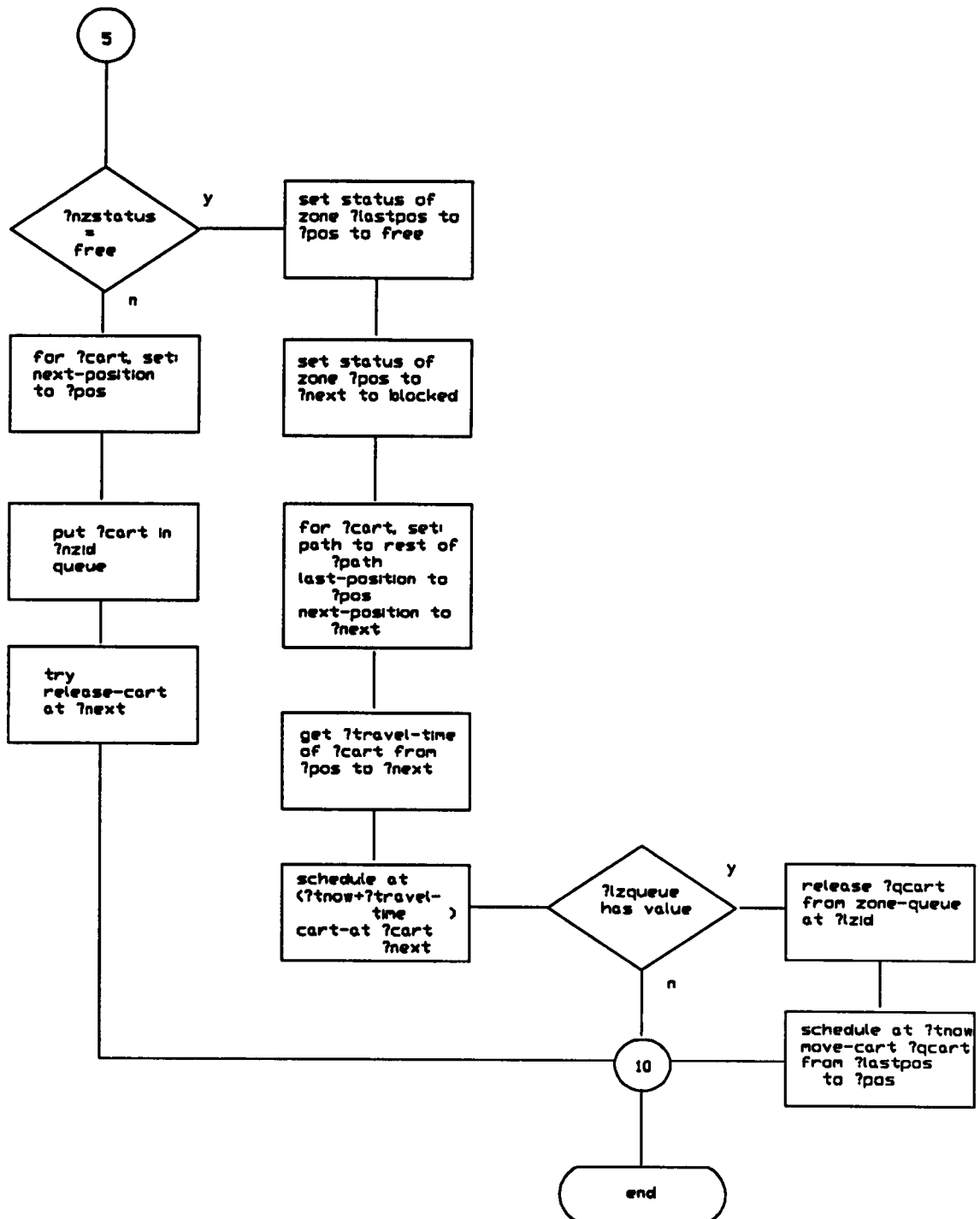
Variables:

?tnow : time move is made.
 ?cart : cart identification code.
 ?pos : movement origin.
 ?to : movement destination.
 ?cpos : cart position.
 ?path : remaining path to final destination.
 ?next : next position in ?path to pass over.
 ?nzid : label of zone between ?pos and ?next.
 ?nzstat : status of zone ?nzid.
 ?lastpos : last known position of cart ?cart.
 ?lzid : label of zone between ?lastpos and ?pos.

Event move-cart

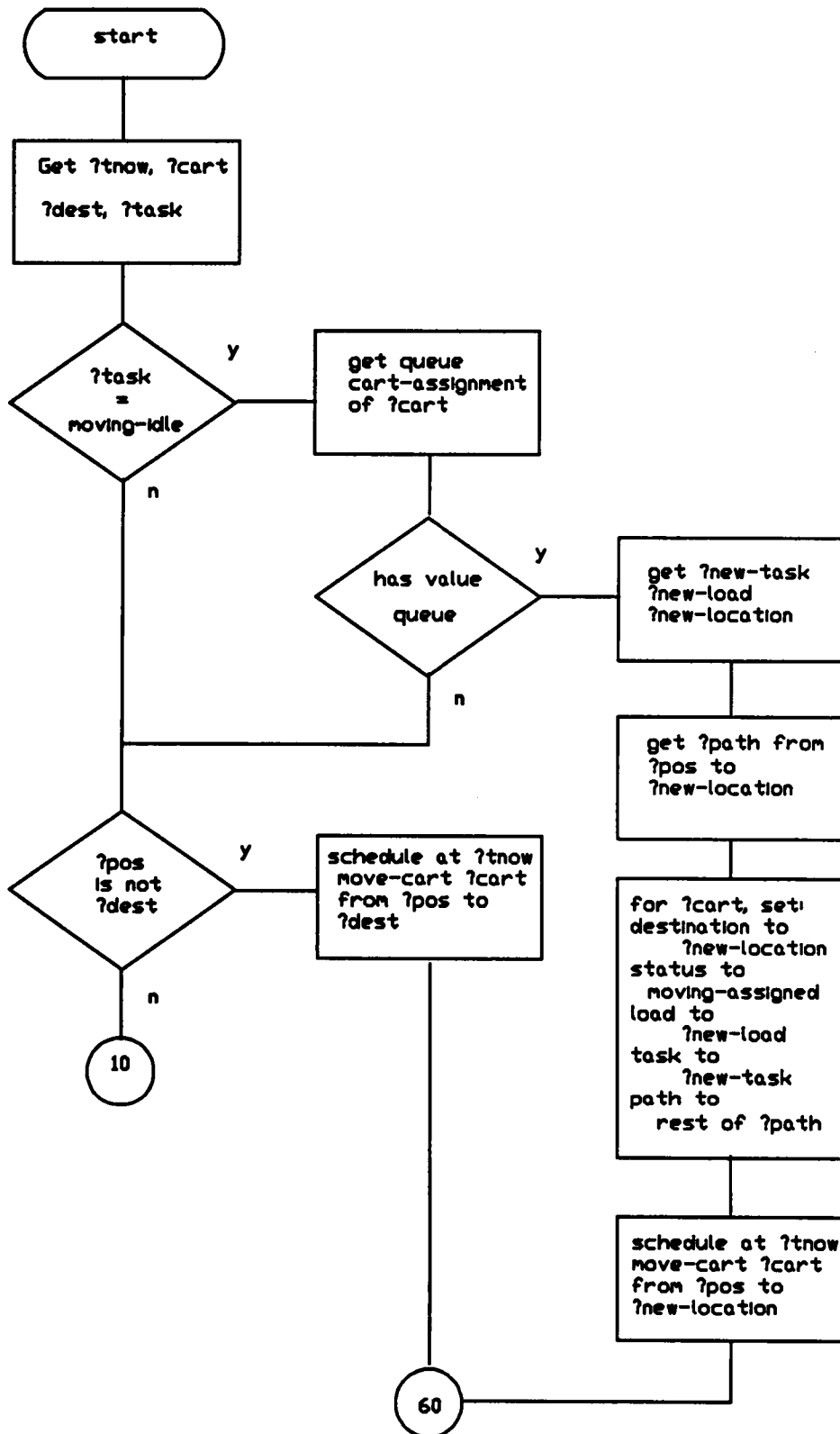
Parms: ?tnow ?cart ?pos ?to

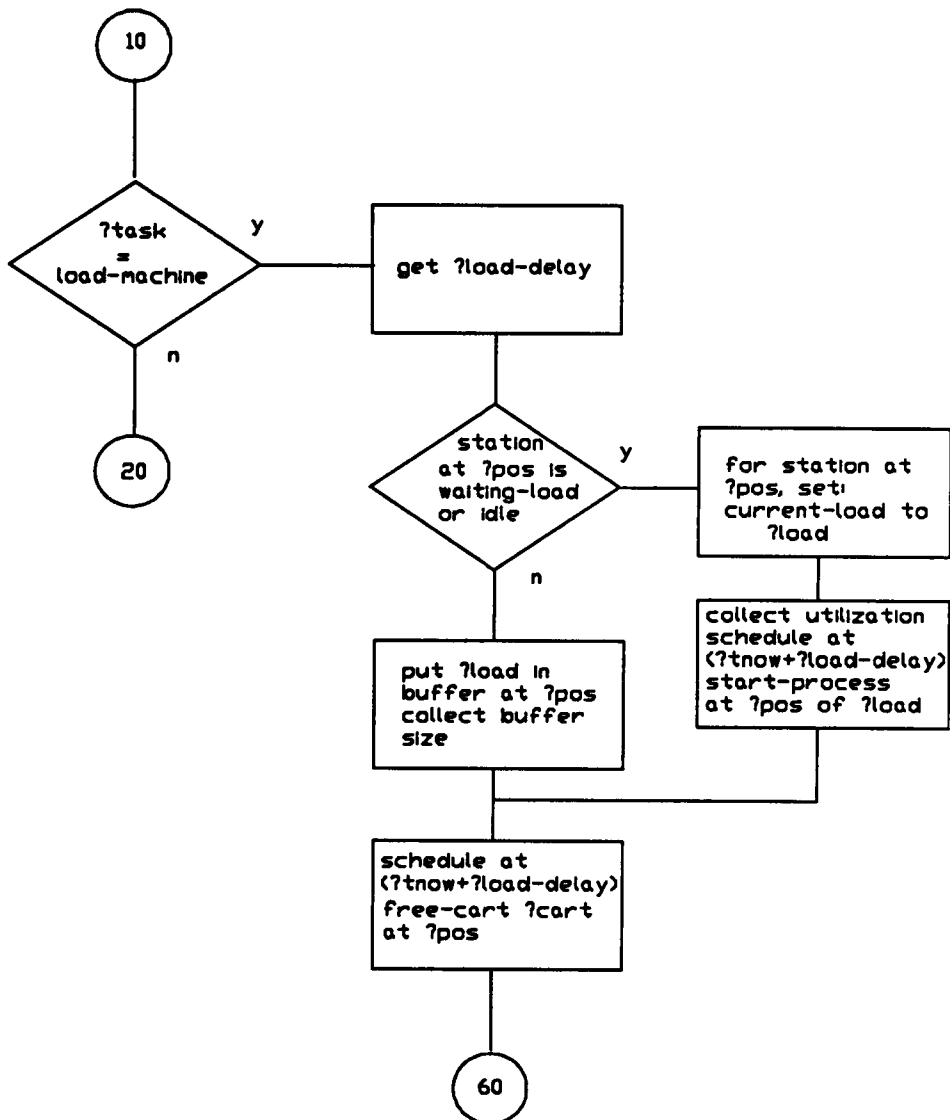


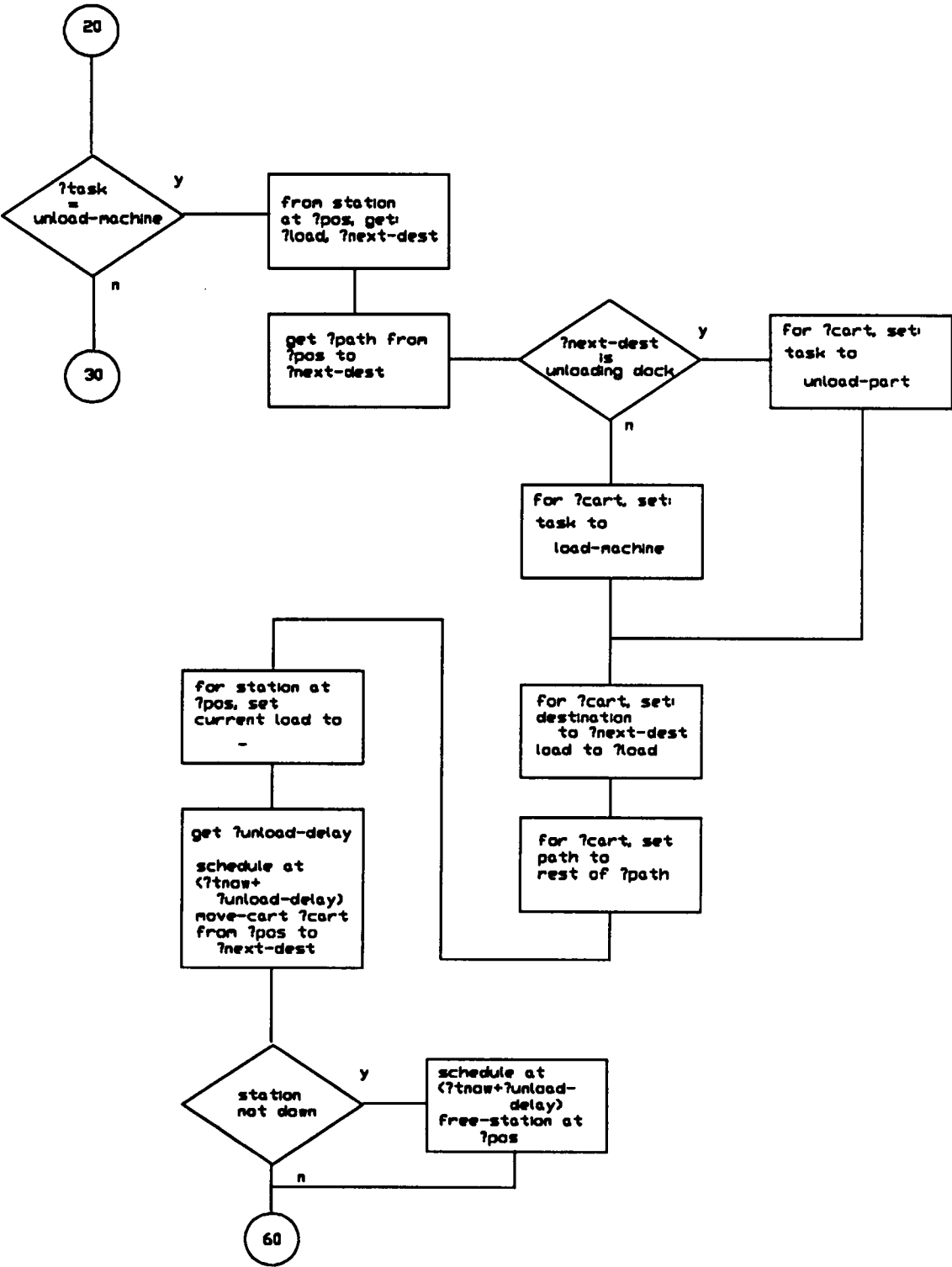


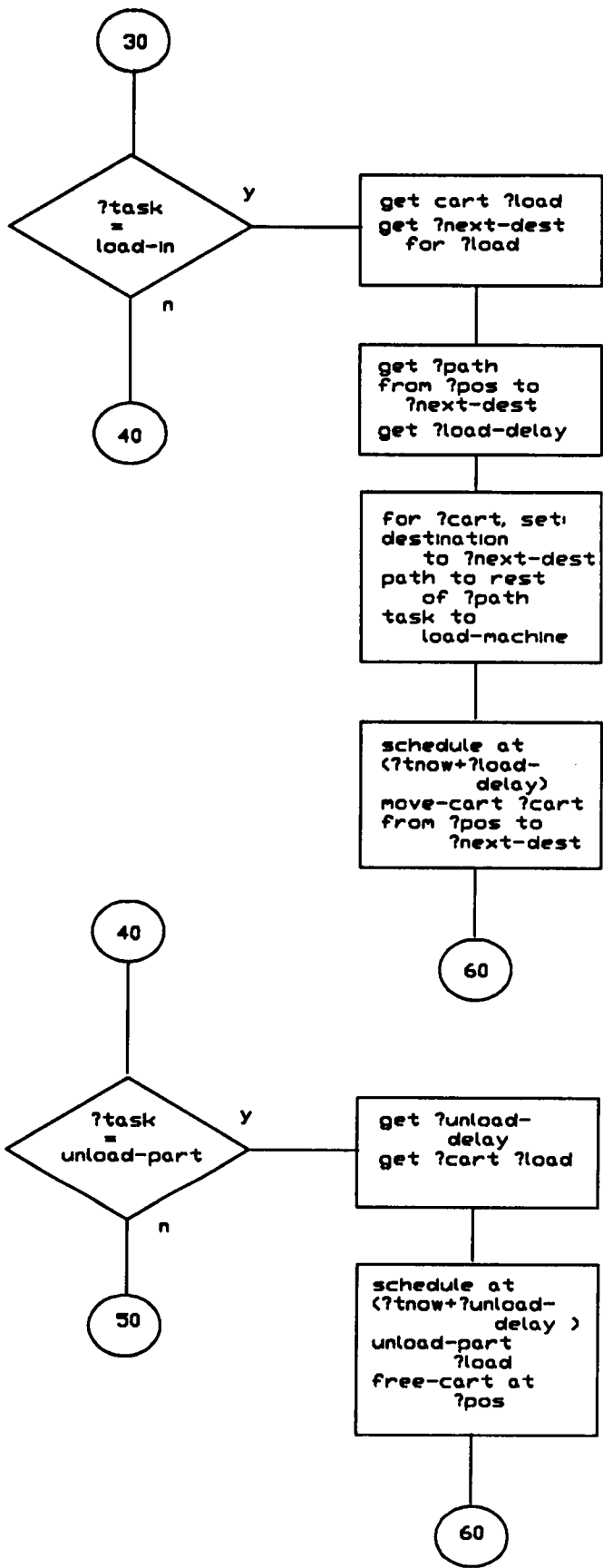
Event Cart-at
Parms: ?tnow ?cart ?pos

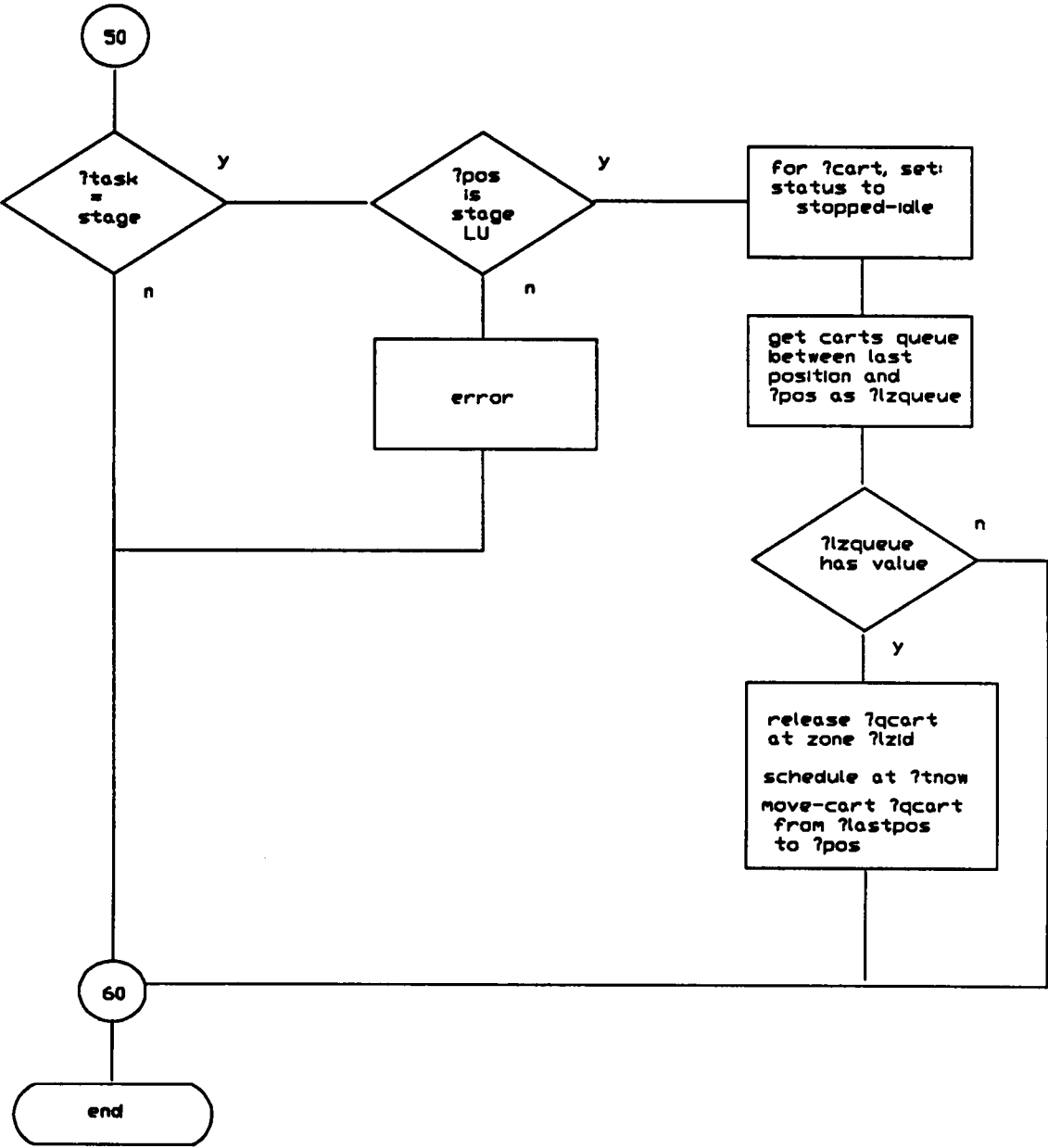
285











APPENDIX F

PROGRAM LISTINGS

The programs used in the research are contained in a 5.25", MS-DOS floppy disk placed on the back cover. This disk contains the files: ' .

CM.LSP	: Main module to run in interactive mode.
FF.LSP	: Forward-chaining inference engine.
FLOOR.LSP	: Objects definition.
FRAMES.LSP	: Frame manipulation functions.
INIT.LSP	: Initialization file for batch mode.
INIT2.LSP	: Initialization file for interactive mode.
KB.LSP	: Knowledge base.
KBA.LSP	: AGV simulation functions.
LIB1.LSP	: Library functions specific to the CM.
LIBRARY.LSP	: General purpose library functions.
MAIN.LSP	: Main module to run in batch mode.
PATTERN.LSP	: Pattern-matching functions.
SEED-SET-1.LSP	: Saved random states for simulation use.
SSIM.LSP	: Simulation interface functions.
UTIL.LSP	: Input-output utility functions.
READ.ME	: Notes on uploading and compilation.

If the floppy disk is not available in this copy, contact:

Dr. William G. Sullivan
Industrial Engineering Dept.
153 Alumni Gym
The University of Tennessee
Knoxville, TN 37996-1506, or

Juan M. Sepulveda
Dept. Ingenieria Industrial
3769 Ave. Ecuador
Universidad de Santiago de Chile
Casilla 10233 Correo-2
Santiago-CHILE

APPENDIX G

DATA TABLES

This appendix contains tables of results for the experiments in Chapter 6.

G.1 Report Example 6.1.

G.2 Report Example 6.3.

G.3 Observed Mean Deviations Given Levels of Mean Repair Times and Time Between Failures.

G.4 Observed Standard Deviations Given Levels of Mean Repair Times and Time Between Failures.

G.5 ANOVA Tables for Section 6.3.1.

G.6 Station Utilization, Station Buffers, Run-Time, and Time in System for Experiments in Section 6.3.1.

G.1
Report

Example 6.1

Time = 588.05

P01 Time	P01 Cm	P01 Model	P02 Time	P02 Cm	P02 Model	P03 Time	P03 Cm	P03 Model	P04 Time	P04 Cm	P04 Model	P05 Time	P05 Cm	P05 Model
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10.00	0.00	0.00	10.00	0.00	0.53	10.00	0.00	0.27	10.00	0.00	0.00	10.00	0.00	0.00
20.00	0.00	0.00	20.00	0.00	1.07	20.00	0.00	0.53	20.00	0.00	0.00	20.00	0.00	0.00
30.00	0.00	0.00	30.00	0.00	1.60	30.00	0.00	0.80	30.00	0.00	0.00	30.00	0.00	0.00
40.00	0.00	0.00	40.00	0.00	2.13	40.00	0.00	1.07	40.00	0.00	0.00	40.00	0.00	0.00
50.00	0.00	0.00	45.85	1.00	2.40	50.00	0.00	1.20	50.00	0.00	0.20	50.00	0.00	0.00
60.00	0.00	0.00	50.00	1.00	2.40	60.00	0.00	1.20	60.00	0.00	0.60	60.00	0.00	0.00
70.00	0.00	0.20	60.00	1.00	2.40	70.00	0.00	1.20	70.00	0.00	0.60	70.00	0.00	0.40
80.00	0.00	0.40	62.40	2.00	2.40	80.00	0.00	1.20	80.00	0.00	0.60	80.00	0.00	0.80
90.00	0.00	0.60	80.00	2.00	2.40	81.40	1.00	1.20	90.00	0.00	0.60	90.00	0.00	1.20
100.00	0.00	0.80	80.00	2.00	2.40	90.00	1.00	1.20	100.00	0.00	0.60	100.00	0.00	1.60
110.00	0.00	0.90	90.00	2.00	2.40	100.00	1.00	1.20	108.00	1.00	0.72	110.00	0.00	1.80
120.00	0.00	0.90	100.00	2.00	2.40	110.00	1.00	1.20	110.00	1.00	0.80	120.00	0.00	1.80
130.00	0.00	0.90	110.00	2.00	2.40	120.00	1.00	1.20	120.00	1.00	1.20	129.55	1.00	1.80
130.80	1.00	0.90	120.60	2.00	2.40	130.00	1.00	1.20	130.00	1.00	1.60	130.00	1.00	1.80
140.00	1.00	0.90	130.00	2.00	2.40	140.00	1.00	1.20	140.00	1.00	2.00	140.00	1.00	1.80
150.00	1.00	0.90	140.00	2.00	2.40	150.00	1.00	1.20	150.00	1.00	2.40	141.10	2.00	1.80
160.00	1.00	1.30	150.00	2.00	2.40	160.00	1.00	1.20	160.00	1.00	2.40	150.00	2.00	1.80
170.00	1.00	1.70	160.00	2.00	2.60	170.00	1.00	1.20	170.00	1.00	2.40	160.00	2.00	1.80
180.00	1.00	2.10	170.00	2.00	2.80	180.00	1.00	1.20	173.35	2.00	2.40	170.00	2.00	1.80
190.00	1.00	2.50	180.00	2.00	3.00	190.00	1.00	1.20	180.00	2.00	2.40	180.00	2.00	1.80
200.00	1.00	2.70	190.00	2.00	3.20	200.00	1.00	1.30	190.00	2.00	2.40	190.00	2.00	1.80
210.00	1.00	2.70	200.00	2.00	3.30	210.00	1.00	1.50	194.35	3.00	2.40	200.00	2.00	1.80
220.00	1.00	2.70	210.00	2.00	3.30	220.00	1.00	1.70	200.00	3.00	2.60	210.00	2.00	1.80
230.00	1.00	2.70	220.00	2.00	3.30	230.00	1.00	1.90	210.00	3.00	3.00	220.00	2.00	1.80
237.35	2.00	2.70	230.00	2.00	3.30	240.00	1.00	2.10	216.90	4.00	3.28	230.00	2.00	1.80
240.00	2.00	2.70	240.00	2.00	3.30	250.00	1.00	2.10	220.00	4.00	3.40	240.00	2.00	1.80
250.00	2.00	2.70	248.90	3.00	3.78	260.00	1.00	2.17	230.00	4.00	3.80	250.00	2.00	1.80
260.00	2.00	2.70	250.00	3.00	3.83	260.55	2.00	2.17	240.00	4.00	4.20	260.00	2.00	2.07
270.00	2.00	2.70	260.00	3.00	4.10	270.00	2.00	2.30	250.00	4.00	4.20	270.00	2.00	2.60
280.00	2.00	2.70	270.00	3.00	4.10	280.00	2.00	2.43	260.00	4.00	4.20	280.00	2.00	3.13
281.45	3.00	2.70	280.00	3.00	4.10	290.00	2.00	2.57	270.00	4.00	4.20	290.00	2.00	3.67
290.00	3.00	2.70	290.00	3.00	4.10	300.00	2.00	2.70	280.00	4.00	4.20	300.00	2.00	4.20
300.00	3.00	2.70	300.00	3.00	4.10	310.00	2.00	2.83	290.00	4.00	4.20	310.00	2.00	4.73
310.00	3.00	2.70	303.50	4.00	4.10	314.50	3.00	2.89	292.45	5.00	4.20	320.00	2.00	5.00
320.00	3.00	2.70	310.00	4.00	4.10	320.00	3.00	2.90	300.00	5.00	4.20	323.95	3.00	5.00

Report Example 6.1

Time = 588.05

P01	P01	P01	P02	P02	P02	P03	P03	P03	P04	P04	P04	P05	P05	P05
Time	Cm	Model	Time	Cm	Model	Time	Cm	Model	Time	Cm	Model	Time	Cm	Model
330.00	3.00	2.70	320.00	4.00	4.37	330.00	3.00	2.90	310.00	5.00	4.20	330.00	3.00	5.00
340.00	3.00	2.70	330.00	4.00	4.90	340.00	3.00	2.90	320.00	5.00	4.20	335.50	4.00	5.00
350.00	3.00	2.70	340.00	4.00	4.90	350.00	3.00	3.03	326.05	6.00	4.20	340.00	4.00	5.00
360.00	3.00	2.70	350.00	4.00	5.17	360.00	3.00	3.30	330.00	6.00	4.20	350.00	4.00	5.00
370.00	3.00	2.70	360.00	4.00	5.70	370.00	3.00	3.57	340.00	6.00	4.60	350.60	5.00	5.00
380.00	3.00	2.70	366.60	5.00	6.05	380.00	3.00	3.83	350.00	6.00	4.80			
390.00	3.00	2.70	370.00	5.00	6.23	390.00	3.00	4.10	360.00	6.00	4.80			
400.00	3.00	2.70	380.00	5.00	6.77	400.00	3.00	4.10	370.00	6.00	4.80			
410.00	3.00	2.70	390.00	5.00	7.30	410.00	3.00	4.10	380.00	6.00	4.80			
420.00	3.00	2.70	400.00	5.00	7.30	420.00	3.00	4.10	390.00	6.00	4.80			
430.00	3.00	2.97	410.00	5.00	7.30	430.00	3.00	4.10	399.75	7.00	5.19			
440.00	3.00	3.23	411.20	6.00	7.30	434.85	4.00	4.10	400.00	7.00	5.20			
450.00	3.00	3.50	420.00	6.00	7.30	440.00	4.00	4.10	410.00	7.00	5.60			
460.00	3.00	3.50	423.30	7.00	7.30	450.00	4.00	4.10	420.00	7.00	6.00			
470.00	3.00	3.66	430.00	7.00	7.30	460.00	4.00	4.10	430.00	7.00	6.27			
480.00	3.00	3.98	440.00	7.00	7.30	470.00	4.00	4.10	440.00	7.00	6.53			
490.00	3.00	4.30	450.00	7.00	7.30	480.00	4.00	4.10	450.00	7.00	6.80			
500.00	3.00	4.46	460.00	7.00	7.30	490.00	4.00	4.10	455.00	8.00	7.00			
500.40	4.00	4.46	470.00	7.00	7.46	500.00	4.00	4.20	460.00	8.00	7.20			
510.00	4.00	4.46	480.00	7.00	7.78	510.00	4.00	4.40	470.00	8.00	7.40			
520.00	4.00	4.46	490.00	7.00	8.10	520.00	4.00	4.40	476.35	9.00	7.40			
530.00	4.00	4.46	500.00	7.00	8.26	530.00	4.00	4.53	480.00	9.00	7.40			
540.00	4.00	4.46	510.00	7.00	8.26	540.00	4.00	4.80	490.00	9.00	7.40			
541.60	5.00	4.46	520.00	7.00	8.26	550.00	4.00	5.07	500.00	9.00	7.60			
			529.50	8.00	8.50	558.10	5.00	5.28	510.00	9.00	8.00			
			530.00	8.00	8.53				512.05	10.00	8.08			
			540.00	8.00	9.06									
			550.00	8.00	9.59									
			560.00	8.00	10.13									
			570.00	8.00	10.66									
			570.85	9.00	10.71									
			580.00	9.00	11.19									
			588.05	10.00	11.62									
Average	2.12	2.39		3.96	4.93		2.20	2.66		4.64	4.10		2.82	3.15

G.2

Report Example 6.3

Time = 605.85

P01 Time	P01 Cm	P01 Model	P02 Time	P02 Cm	P02 Model	P03 Time	P03 Cm	P03 Model	P04 Time	P04 Cm	P04 Model	P05 Time	P05 Cm	P05 Model
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
10.00	0.00	0.00	10.00	0.00	0.53	10.00	0.00	0.27	10.00	0.00	0.00	10.00	0.00	0.00
20.00	0.00	0.00	20.00	0.00	1.07	20.00	0.00	0.53	20.00	0.00	0.00	20.00	0.00	0.00
30.00	0.00	0.00	30.00	0.00	1.60	30.00	0.00	0.80	30.00	0.00	0.00	30.00	0.00	0.00
40.00	0.00	0.00	40.00	0.00	2.13	40.00	0.00	1.07	40.00	0.00	0.00	40.00	0.00	0.00
50.00	0.00	0.00	45.85	1.00	2.40	50.00	0.00	1.20	50.00	0.00	0.20	50.00	0.00	0.00
60.00	0.00	0.00	50.00	1.00	2.40	60.00	0.00	1.20	60.00	0.00	0.60	60.00	0.00	0.00
70.00	0.00	0.20	60.00	1.00	2.40	70.00	0.00	1.20	70.00	0.00	0.60	70.00	0.00	0.40
80.00	0.00	0.40	62.40	2.00	2.40	80.00	0.00	1.20	80.00	0.00	0.60	80.00	0.00	0.80
90.00	0.00	0.60	70.00	2.00	2.40	81.40	1.00	1.20	90.00	0.00	0.60	90.00	0.00	1.20
100.00	0.00	0.80	80.00	2.00	2.40	90.00	1.00	1.20	100.00	0.00	0.60	100.00	0.00	1.60
110.00	0.00	0.90	90.00	2.00	2.40	100.00	1.00	1.20	108.00	1.00	0.72	110.00	0.00	1.80
120.00	0.00	0.90	100.00	2.00	2.40	110.00	1.00	1.20	110.00	1.00	0.80	120.00	0.00	1.80
130.00	0.00	0.90	110.00	2.00	2.40	120.00	1.00	1.20	120.00	1.00	1.20	129.55	1.00	1.80
130.80	1.00	0.90	120.00	2.00	2.40	130.00	1.00	1.20	130.00	1.00	1.60	130.00	1.00	1.80
140.00	1.00	0.90	130.00	2.00	2.40	140.00	1.00	1.20	140.00	1.00	2.00	140.00	1.00	1.80
150.00	1.00	0.90	140.00	2.00	2.40	150.00	1.00	1.20	150.00	1.00	2.40	141.10	2.00	1.80
160.00	1.00	1.30	150.00	2.00	2.40	160.00	1.00	1.20	160.00	1.00	2.40	150.00	2.00	1.80
170.00	1.00	1.70	160.00	2.00	2.60	170.00	1.00	1.20	170.00	1.00	2.40	160.00	2.00	1.80
180.00	1.00	2.10	170.00	2.00	2.80	180.00	1.00	1.20	173.35	2.00	2.40	170.00	2.00	1.80
190.00	1.00	2.50	180.00	2.00	3.00	190.00	1.00	1.20	180.00	2.00	2.40	180.00	2.00	1.80
200.00	1.00	2.70	190.00	2.00	3.20	200.00	1.00	1.30	190.00	2.00	2.40	190.00	2.00	1.80
210.00	1.00	2.70	200.00	2.00	3.30	210.00	1.00	1.50	194.35	3.00	2.40	200.00	2.00	1.80
220.00	1.00	2.70	210.00	2.00	3.30	220.00	1.00	1.63	200.00	3.00	2.60	210.00	2.00	1.80
230.00	1.00	2.70	220.00	2.00	3.30	230.00	1.00	1.63	210.00	3.00	3.00	220.00	2.00	1.80
237.35	2.00	2.70	230.00	2.00	3.30	240.00	1.00	1.82	216.90	4.00	3.25	230.00	2.00	1.80
240.00	2.00	2.70	240.00	2.00	3.68	250.00	1.00	2.02	220.00	4.00	3.25	240.00	2.00	1.80
250.00	2.00	2.70	248.90	3.00	3.68	260.00	1.00	2.22	230.00	4.00	3.25	250.00	2.00	1.80
260.00	2.00	2.70	250.00	3.00	3.68	260.55	2.00	2.23	240.00	4.00	3.25	260.00	2.00	1.80
270.00	2.00	2.70	260.00	3.00	3.68	270.00	2.00	2.42	250.00	4.00	3.65	270.00	2.00	1.80
280.00	2.00	2.70	270.00	3.00	3.68	280.00	2.00	2.62	260.00	4.00	4.05	280.00	2.00	1.80
280.90	3.00	2.70	280.00	3.00	3.68	290.00	2.00	2.72	270.00	4.00	4.45	290.00	2.00	1.80
290.00	3.00	2.70	290.00	3.00	3.95	300.00	3.00	2.72	280.00	4.00	4.85	300.00	2.00	1.80
300.00	3.00	2.70	300.00	3.00	4.48	303.45	3.00	2.72	290.00	4.00	5.05	310.00	2.00	2.33
310.00	3.00	2.70	310.00	3.00	4.48	310.00	3.00	2.72	300.00	4.00	5.05	320.00	2.00	2.60
320.00	3.00	2.70	320.00	3.00	4.75	320.00	3.00	2.72	310.00	4.00	5.05	330.00	2.00	2.60

Report Example 6.3

Time = 605.85

P01	P01	P01	P02	P02	P02	P03	P03	P03	P04	P04	P04	P05	P05	P05
Time	Cm	Model	Time	Cm	Model	Time	Cm	Model	Time	Cm	Model	Time	Cm	Model
330.00	3.00	2.70	330.00	3.00	5.28	330.00	3.00	2.72	315.00	5.00	5.05	340.00	2.00	2.60
340.00	3.00	2.70	340.00	3.00	5.28	340.00	3.00	2.92	320.00	5.00	5.05	350.00	2.00	2.60
350.00	3.00	2.70	350.00	3.00	5.28	350.00	3.00	3.12	326.55	6.00	5.05	360.00	2.00	2.60
360.00	3.00	2.70	353.10	4.00	5.28	360.00	3.00	3.32	330.00	6.00	5.05	370.00	2.00	2.60
370.00	3.00	2.70	360.00	4.00	5.28	370.00	3.00	3.52	340.00	6.00	5.45	380.00	2.00	2.60
380.00	3.00	2.70	370.00	4.00	5.28	380.00	3.00	3.62	341.55	7.00	5.51	380.55	3.00	2.60
390.00	3.00	2.70	379.30	5.00	5.51	390.00	3.00	3.62	350.00	7.00	5.85	390.00	3.00	2.60
400.00	3.00	2.90	380.00	5.00	5.55	391.50	4.00	3.62	360.00	7.00	6.25	400.00	3.00	3.00
410.00	3.00	3.10	390.00	5.00	6.08	400.00	4.00	3.62	370.00	7.00	6.65	410.00	3.00	3.40
420.00	3.00	3.30	400.00	5.00	6.08	410.00	4.00	3.62	380.00	7.00	6.85	420.00	3.00	3.80
430.00	3.00	3.50	410.00	5.00	6.08	420.00	4.00	3.62	390.00	7.00	6.85	430.00	3.00	4.20
440.00	3.00	3.57	420.00	5.00	6.08	430.00	4.00	3.62	400.00	7.00	6.85	440.00	3.00	4.34
450.00	3.00	3.57	430.00	5.00	6.08	440.00	4.00	3.62	410.00	7.00	6.85	450.00	3.00	4.34
460.00	3.00	3.57	440.00	5.00	6.08	450.00	4.00	3.68	413.10	8.00	6.85	460.00	3.00	4.34
470.00	3.00	3.57	445.75	6.00	6.08	460.00	4.00	3.95	420.00	8.00	6.85	470.00	3.00	4.34
474.35	4.00	3.57	450.00	6.00	6.21	470.00	4.00	4.21	430.00	8.00	6.85	473.10	4.00	4.34
480.00	4.00	3.57	460.00	6.00	6.74	480.00	4.00	4.48	434.10	9.00	6.85	480.00	4.00	4.34
490.00	4.00	3.57	470.00	6.00	7.27	490.00	4.00	4.75	440.00	9.00	6.85	485.30	5.00	4.34
500.00	4.00	3.57	480.00	6.00	7.81	500.00	4.00	4.88	450.00	9.00	6.85			
510.00	4.00	3.57	490.00	6.00	8.34	510.00	4.00	4.88	460.00	9.00	6.85			
520.00	4.00	3.84	500.00	6.00	8.61	520.00	4.00	4.88	470.00	9.00	6.85			
530.00	4.00	3.97	500.25	7.00	8.61	530.00	4.00	4.88	480.00	9.00	6.85			
540.00	4.00	3.97	510.00	7.00	8.61	531.40	5.00	4.88	490.00	9.00	6.85			
550.00	4.00	4.33	516.80	8.00	8.61				500.00	9.00	7.05			
560.00	4.00	4.68	520.00	8.00	8.61				510.00	9.00	7.45			
570.00	4.00	5.04	530.00	8.00	8.61				520.00	9.00	7.72			
580.00	4.00	5.39	540.00	8.00	8.61				530.00	9.00	8.05			
581.80	5.00	5.46	550.00	8.00	8.78				540.00	9.00	8.45			
			560.00	8.00	8.96				550.00	9.00	8.54			
			570.00	8.00	9.14				560.00	9.00	8.63			
			580.00	8.00	9.32				560.80	10.00	8.64			
			590.00	8.00	9.50									
			593.75	9.00	9.56									
			600.00	9.00	9.67									
			605.85	10.00	9.78									

Avg.

2.19 2.47

3.81 4.82

2.41 2.68

4.91 4.49

2.34 2.47

OBSERVED MEAN DEVIATIONS GIVEN LEVELS OF MEAN
REPAIR TIMES AND TIME BETWEEN FAILURES

TR: REPAIR TIME (min) Uniform			TBF: TIME BETWEEN FAILURES ALL STATIONS (min) Exponential	
Lower	Upper	Mean RT (MTR)	Mean TBF (MTBF)	
10	20	15	60	
20	40	30	120	
50	70	60	240	
110	130	120	480	

MRT	MTBF	D1	D2	D3	D4	D5
15	480	-0.47	0.28	-0.31	-0.61	1.25
15	480	-0.75	1.68	0.45	-0.84	0.24
15	480	-0.13	0.75	-0.15	-0.88	0.72
15	480	-0.16	0.35	0.06	-0.58	0.72
15	480	-0.30	0.75	1.29	-0.40	-0.34
15	240	-1.16	-0.06	-0.25	1.03	0.49
15	240	-0.14	0.71	-0.03	0.23	0.38
15	240	-0.68	0.28	1.05	0.03	-0.48
15	240	-1.02	-1.15	-0.78	3.12	0.53
15	240	-1.37	0.12	1.20	0.65	0.18
15	120	-0.83	-0.71	-0.34	1.21	0.18
15	120	-1.17	-0.01	-0.06	0.69	0.58
15	120	-0.92	-2.06	-0.82	2.89	0.44
15	120	-0.92	-0.19	0.84	-0.10	-0.38
15	120	-0.84	-0.40	0.92	0.12	-0.39
15	60	-0.40	-0.47	-0.41	1.93	-0.22
15	60	-1.22	2.14	-1.37	-0.92	0.69
15	60	-1.10	0.62	-0.58	-0.07	1.00
15	60	-0.66	-1.04	-0.87	1.31	-0.95
15	60	-1.95	-1.81	0.44	3.07	-0.40
30	480	0.10	0.83	-0.06	-0.64	0.06
30	480	-0.60	-0.27	0.09	0.53	-0.46
30	480	0.41	-0.21	-0.28	0.11	1.18
30	480	0.23	0.31	-0.43	0.13	0.72
30	480	-0.65	1.23	-0.11	-0.23	0.85
30	240	-0.85	0.23	0.66	1.47	-0.33
30	240	-0.22	2.05	-0.45	-1.18	0.13
30	240	-0.94	1.46	-0.62	0.82	0.03
30	240	-0.35	0.39	0.03	0.29	0.93
30	240	-0.12	0.29	1.11	1.48	-0.26

MRT	MTBF	D1	D2	D3	D4	D5
30	120	-1.08	-0.21	0.27	0.71	1.64
30	120	-0.67	0.01	-0.06	-0.09	0.79
30	120	0.42	1.05	0.03	0.05	-0.68
30	120	-0.40	-0.27	0.40	-0.61	1.39
30	120	-0.64	0.73	-0.01	0.82	-0.02
30	60	-0.85	-1.28	-0.07	1.31	0.50
30	60	-1.21	-1.17	-0.44	0.06	-0.64
30	60	-1.45	-0.99	1.36	0.60	0.35
30	60	-1.45	-2.31	0.69	1.50	-0.40
30	60	-1.13	-0.80	-0.49	0.24	-0.20
60	480	-0.73	0.47	1.11	-0.26	0.54
60	480	-0.93	1.18	0.49	-0.16	0.51
60	480	0.41	0.16	-0.27	0.29	-0.21
60	480	-1.63	0.45	-0.13	1.07	0.43
60	480	-0.09	0.65	-0.07	0.04	0.50
60	240	-0.44	-0.89	0.40	0.31	1.09
60	240	-0.59	0.89	0.01	-0.41	-0.15
60	240	-0.17	2.00	-0.12	0.07	-1.05
60	240	0.19	-0.57	0.64	1.57	0.05
60	240	-1.35	0.24	1.22	0.47	0.23
60	120	-1.09	-0.15	0.73	1.93	-0.41
60	120	-0.55	0.72	-0.82	-0.02	1.30
60	120	-1.51	0.54	-0.01	2.11	1.00
60	120	-0.83	0.92	0.01	-0.54	0.76
60	120	-1.33	-0.57	-0.46	2.24	0.23
60	60	-0.96	-0.46	-0.51	-0.39	-1.02
60	60	-1.18	-1.78	1.40	0.77	1.18
60	60	-0.27	-0.97	0.19	0.63	-0.22
60	60	-1.39	0.45	-0.14	-1.52	1.01
60	60	-1.32	-0.82	1.81	1.52	-0.55
120	480	-0.74	1.29	0.42	0.64	0.65
120	480	-0.20	1.01	0.42	0.26	-0.17
120	480	-0.42	1.12	0.74	-0.37	1.50
120	480	-0.37	0.75	1.59	0.33	1.17
120	480	-0.47	1.59	-0.05	-0.99	1.24
120	240	-1.50	1.58	0.64	0.47	1.39
120	240	-0.56	0.42	0.93	0.74	0.31
120	240	-1.29	-0.46	-0.13	2.20	1.34
120	240	-1.07	0.17	0.70	-1.41	2.39
120	240	-0.14	-0.28	0.70	0.07	-0.25

MRT	MTBF	D1	D2	D3	D4	D5
120	120	-0.91	-0.09	-1.20	0.37	-0.16
120	120	-0.42	0.19	-0.02	0.65	0.85
120	120	-0.78	0.51	0.24	-0.18	-0.09
120	120	0.31	-0.07	0.00	0.44	-0.39
120	120	-0.98	-0.70	0.50	-0.83	1.29

G.4

OBSERVED STANDARD DEVIATIONS GIVEN LEVELS OF MEAN
REPAIR TIMES AND TIME BETWEEN FAILURES
(Based on S)
 $n-1$

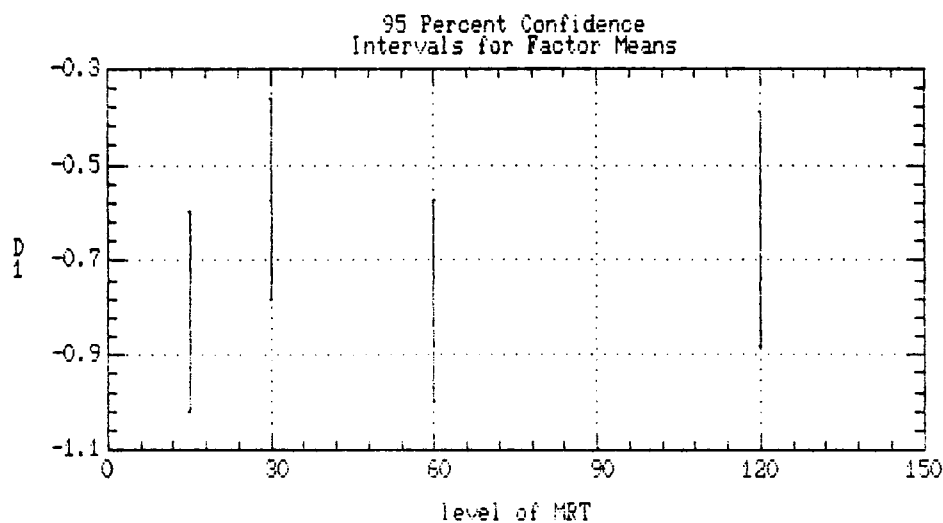
MRT	MTBF	SDD1	SDD2	SDD3	SDD4	SDD5
15	480	0.45	0.71	0.66	0.47	0.62
15	480	0.55	0.96	0.56	0.83	0.62
15	480	0.26	0.54	0.87	0.70	0.78
15	480	0.50	0.64	0.34	0.72	0.88
15	480	0.62	0.62	0.20	0.81	0.88
MEAN		0.48	0.69	0.53	0.71	0.76
30	480	0.53	0.84	0.62	0.88	0.88
30	480	0.71	0.58	0.58	0.74	0.81
30	480	0.71	0.80	0.07	0.50	0.25
30	480	0.81	0.64	0.21	0.89	0.36
30	480	0.48	0.48	0.25	1.06	0.56
MEAN		0.65	0.67	0.35	0.81	0.57
60	480	0.52	1.05	0.51	1.04	0.71
60	480	0.73	0.68	0.31	0.56	0.87
60	480	0.54	0.51	0.41	0.83	0.82
60	480	0.23	0.70	0.30	0.64	0.84
60	480	0.66	0.70	0.61	0.79	0.59
MEAN		0.54	0.73	0.43	0.77	0.77
120	480	0.68	0.75	0.54	1.26	0.90
120	480	0.58	0.91	0.49	0.74	1.18
120	480	0.40	0.68	0.27	0.63	1.21
120	480	0.91	0.95	0.49	0.54	0.80
120	480	0.34	0.64	0.67	0.73	1.12
MEAN		0.58	0.79	0.49	0.78	1.04
15	240	1.01	0.79	0.49	1.24	1.03
15	240	0.58	0.92	0.78	0.60	0.88
15	240	0.25	0.52	0.21	1.15	0.58
15	240	0.80	0.54	0.44	0.85	0.79
15	240	0.36	0.64	0.47	0.43	0.51
MEAN		0.60	0.68	0.48	0.85	0.76

MRT	MTBF	SDD1	SDD2	SDD3	SDD4	SDD5
30	240	0.65	0.56	0.77	0.72	0.79
30	240	0.74	0.69	0.20	0.66	0.77
30	240	0.51	0.64	0.17	0.77	1.21
30	240	0.34	1.09	0.51	1.13	0.53
30	240	0.62	0.96	0.69	0.92	0.40
MEAN		0.57	0.79	0.47	0.84	0.74
60	240	0.82	0.85	0.60	0.79	0.62
60	240	0.80	0.77	0.52	1.09	1.24
60	240	0.42	0.67	0.73	1.17	0.81
60	240	0.71	0.74	1.12	0.84	0.56
60	240	0.53	1.08	0.37	1.03	0.79
MEAN		0.66	0.82	0.67	0.98	0.80
120	240	0.88	0.76	0.54	1.21	0.79
120	240	0.77	0.45	0.71	0.58	0.39
120	240	0.83	0.87	0.42	1.03	0.69
120	240	0.63	1.23	0.50	0.61	0.93
120	240	0.68	0.60	0.64	1.21	1.43
MEAN		0.76	0.78	0.56	0.93	0.85
15	120	0.46	0.69	0.58	0.45	0.66
15	120	0.21	0.88	0.67	0.71	1.20
15	120	0.39	0.61	0.60	0.85	0.45
15	120	0.62	1.00	0.88	0.92	0.49
15	120	0.68	1.03	0.39	0.61	0.34
MEAN		0.47	0.84	0.62	0.71	0.63
30	120	0.70	0.84	0.42	1.32	0.68
30	120	0.75	0.52	0.17	0.52	0.51
30	120	0.66	0.96	0.19	1.37	0.52
30	120	0.71	0.53	0.29	1.01	0.43
30	120	0.54	0.82	0.34	0.79	1.10
MEAN		0.67	0.73	0.28	1.00	0.65
60	120	1.10	1.17	0.51	1.12	0.60
60	120	0.67	1.31	0.65	1.17	0.80
60	120	1.03	0.79	0.41	0.85	1.08
60	120	1.21	1.05	1.11	1.12	0.60
60	120	0.97	0.70	0.86	0.88	0.79
MEAN		1.00	1.00	0.71	1.03	0.77

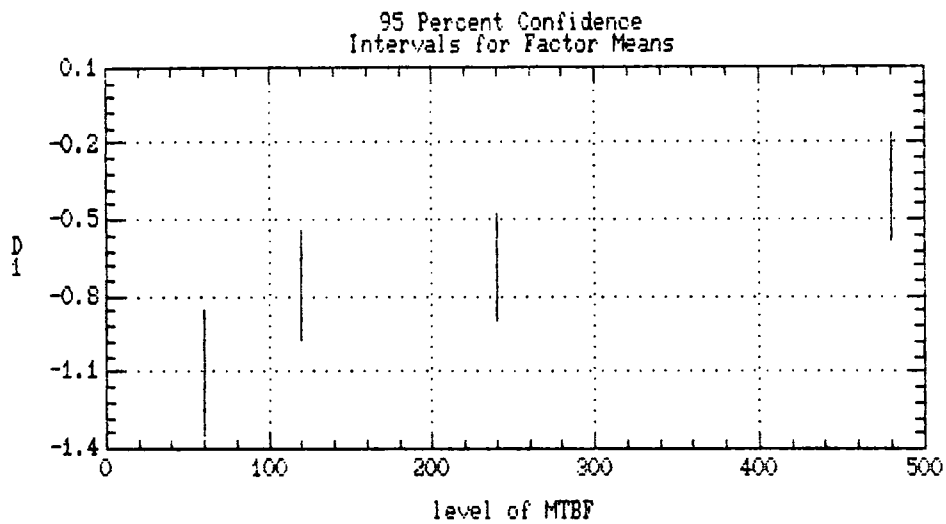
MRT	MTBF	SDD1	SDD2	SDD3	SDD4	SDD5
120	120	0.70	0.71	1.01	0.59	0.84
120	120	0.90	0.87	1.10	0.65	0.92
120	120	0.52	1.15	0.49	1.21	1.13
120	120	0.78	0.59	0.95	0.93	0.86
120	120	0.96	0.96	0.69	1.05	0.74
=====						
MEAN		0.77	0.86	0.85	0.89	0.90
15	60	0.50	0.51	0.68	0.85	0.46
15	60	0.98	1.54	1.05	1.50	0.52
15	60	1.06	0.73	0.52	0.56	0.40
15	60	0.49	0.93	0.63	0.75	0.64
15	60	0.79	1.15	0.73	0.84	0.39
=====						
MEAN		0.76	0.97	0.72	0.90	0.48
30	60	1.16	0.80	1.15	1.71	0.64
30	60	0.88	1.02	0.42	0.73	0.29
30	60	1.10	1.56	0.31	0.95	1.00
30	60	0.57	1.46	0.50	0.90	0.64
30	60	0.77	0.72	0.55	0.90	0.47
=====						
MEAN		0.90	1.11	0.59	1.04	0.61
60	60	1.20	1.50	0.92	1.40	0.92
60	60	0.74	0.76	0.87	1.06	0.46
60	60	0.93	0.59	0.77	0.81	0.68
60	60	0.94	0.99	1.06	1.01	0.43
60	60	0.68	0.65	0.62	0.78	0.73
=====						
MEAN		0.90	0.90	0.85	1.01	0.64

RESPONSE VARIABLE: D1					
SOURCE OF VARIATION	SUM OF SQUARES	D.F.	MEAN SQUARE	F-RATIO	PROB(>F)
MAIN EFFECTS	5.3205076	6	.8867513	3.9737782	.0021
MRT	.7043522	3	.2347841	1.0521325	.3762
MTBF	4.5418389	3	1.5139463	6.7844131	.0005
2-FACTOR INTERACTIONS	2.1034711	8	.2629339	1.1782796	.3272
MRT MTBF	2.1034711	8	.2629339	1.1782796	.3272
RESIDUAL	13.389040	60	.2231507		
TOTAL (CORR.)	20.813019	74			

Press ENTER to continue.

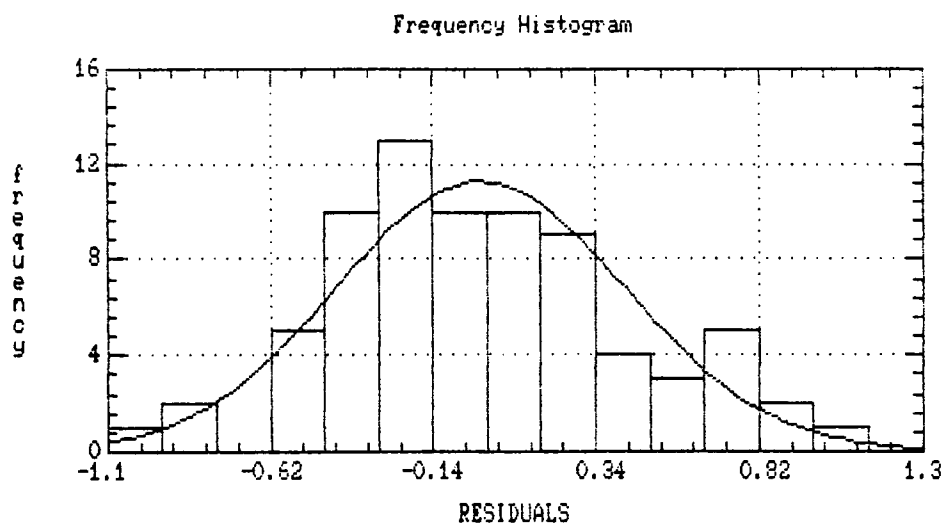


Press ENTER to continue.



LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
60	15	-.396800	-1.102667	.121970	-1.3466	-0.8587
120	20	-.051133	-.757000	.105629	-0.9683	-0.5457
240	20	.017367	-.688500	.105629	-0.8998	-0.4772
480	20	.331367	-.374500	.105629	-0.5858	-0.1632

Press ENTER to continue.



ESTIMATED PARAMETERS: 6.3653E-17 0.42536
 CHI² GOODNESS-OF-FIT STATISTIC = 4.5945 WITH 6 DEGREES OF FREEDOM
 PROBABILITY OF A LARGER VALUE = 0.59677

***** S T A T G R A P H I C S *****

Onesample Analysis

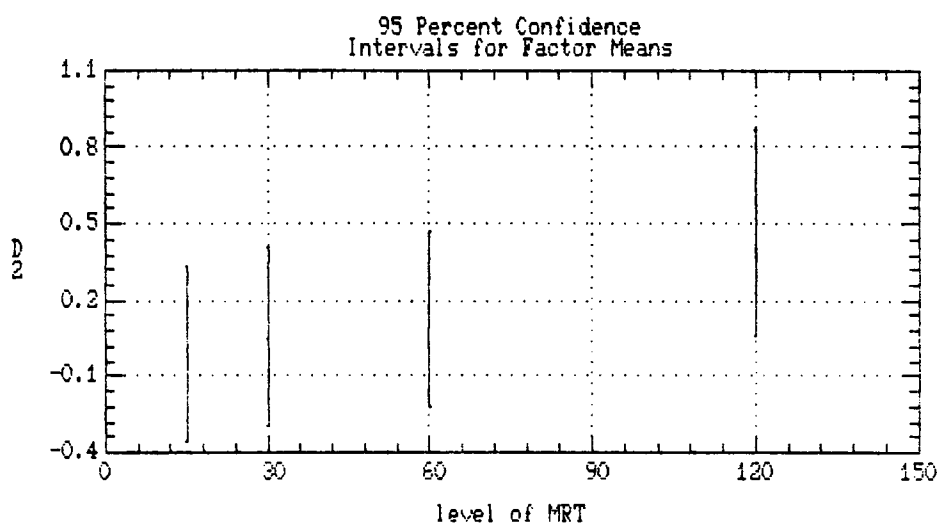
RESIDUALS

Sample Statistics:	Number of Obs.	75
	Average	6.0187E-17
	Variance	0.18093
	Std. Deviation	0.42536
	Median	8E-3
Confidence Interval for Mean:	95 Percent	
Sample 1	-0.097889 0.097889	74 D.F.
Confidence Interval for Variance:	0 Percent	
Sample 1		

Hypothesis Test for H0: Mean = 0	Computed T statistic = 1.2254E-15
vs Alt: NE	Sig. Level = 1
at Alpha = 0.05	so do not reject H0.

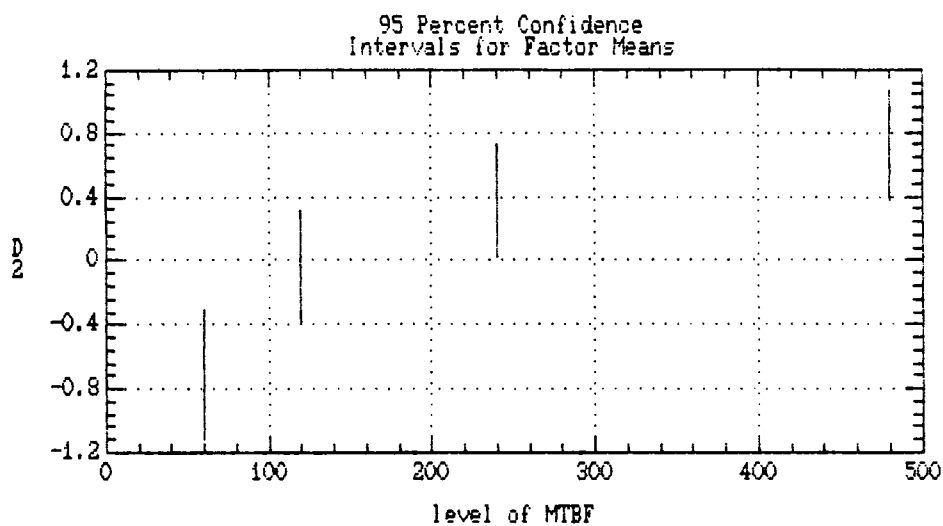
RESPONSE VARIABLE: D2					
SOURCE OF VARIATION	SUM OF SQUARES	D.F.	MEAN SQUARE	F-RATIO	PROB(>F)
MAIN EFFECTS	19.758321	6	3.2930535	5.4151910	.0002
MRT	.4589106	3	.1529702	.2515485	.8599
MTBF	17.526891	3	5.8422969	9.6072395	.0000
2-FACTOR INTERACTIONS	9.8914978	8	1.2364372	2.0332326	.0574
MRT MTBF	9.8914978	8	1.2364372	2.0332326	.0574
RESIDUAL	36.486840	60	.6081140		
TOTAL (CORR.)	66.136659	74			

Press ENTER to continue.



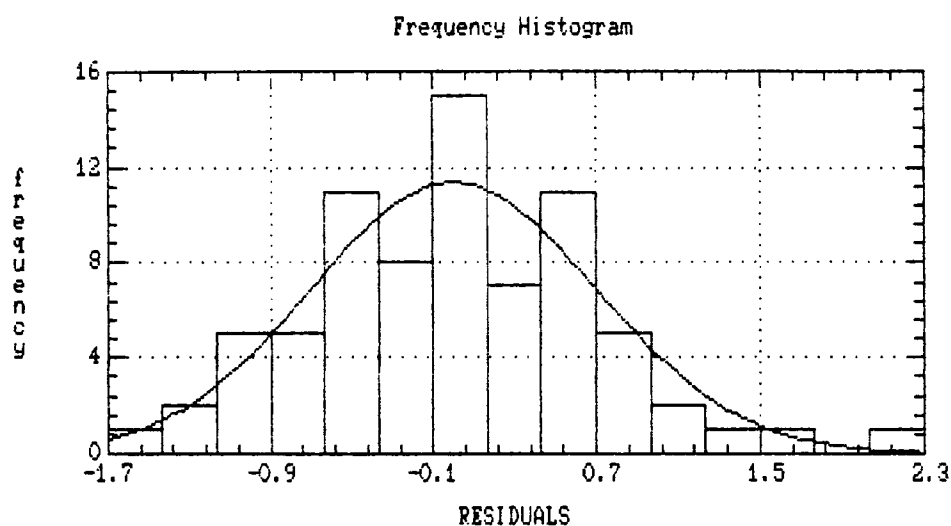
LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
15	20	-.148867	-.011000	.174372	-0.3597	0.3377
30	20	-.084367	.053500	.174372	-0.2952	0.4022
60	20	-.014867	.123000	.174372	-0.2257	0.4717
120	15	.330800	.468667	.201348	0.0660	0.8714

Press ENTER to continue.



LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
60	15	-.850533	-.712667	.201348	-1.1154	-0.3100
120	20	-.175867	-.038000	.174372	-0.3867	0.3107
240	20	.233133	.371000	.174372	0.0223	0.7197
480	20	.580633	.718500	.174372	0.3698	1.0672

Press ENTER to continue.



ESTIMATED PARAMETERS: 3.1456E-18 0.70219
 CHI² GOODNESS-OF-FIT STATISTIC = 5.2521 WITH 6 DEGREES OF FREEDOM
 PROBABILITY OF A LARGER VALUE = 0.51191

***** STATGRAPHICS *****

Onesample Analysis

RESIDUALS

Sample Statistics: Number of Obs. 75
 Average 1.1384E-18
 Variance 0.49307
 Std. Deviation 0.70219
 Median -0.036

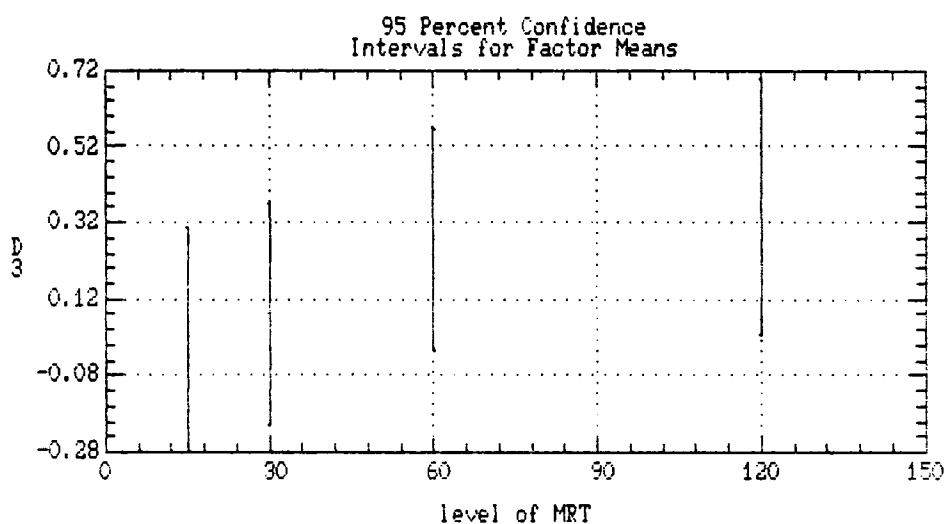
Confidence Interval for Mean: 95 Percent
 Sample 1 -0.16159 0.16159 74 D.F.

Confidence Interval for Variance: 0 Percent
 Sample 1

Hypothesis Test for H₀: Mean = 0 Computed T statistic = 1.404E-17
 vs Alt: NE Sig. Level = 1
 at Alpha = 0.05 so do not reject H₀.

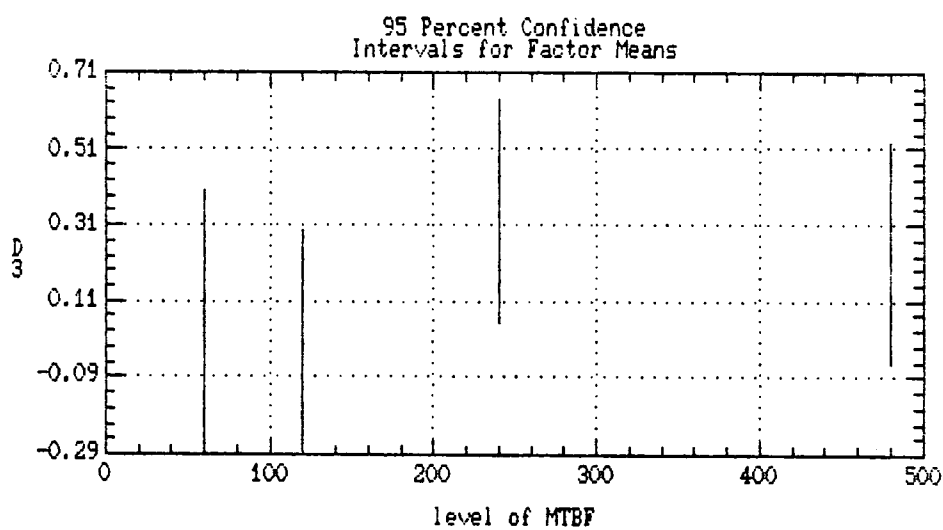
RESPONSE VARIABLE: D3					
SOURCE OF VARIATION	SUM OF SQUARES	D.F.	MEAN SQUARE	F-RATIO	PROB(>F)
MAIN EFFECTS	2.6956109	6	.4492685	1.0640256	.3941
MRT	1.2922806	3	.4307602	1.0201914	.3901
MTBF	1.2619856	3	.4206619	.9962750	.4008
2-FACTOR INTERACTIONS	4.2470478	8	.5308810	1.2573126	.2829
MRT MTBF	4.2470478	8	.5308810	1.2573126	.2829
RESIDUAL	25.334080	60	.4222347		
TOTAL (CORR.)	32.276739	74			

Press ENTER to continue.



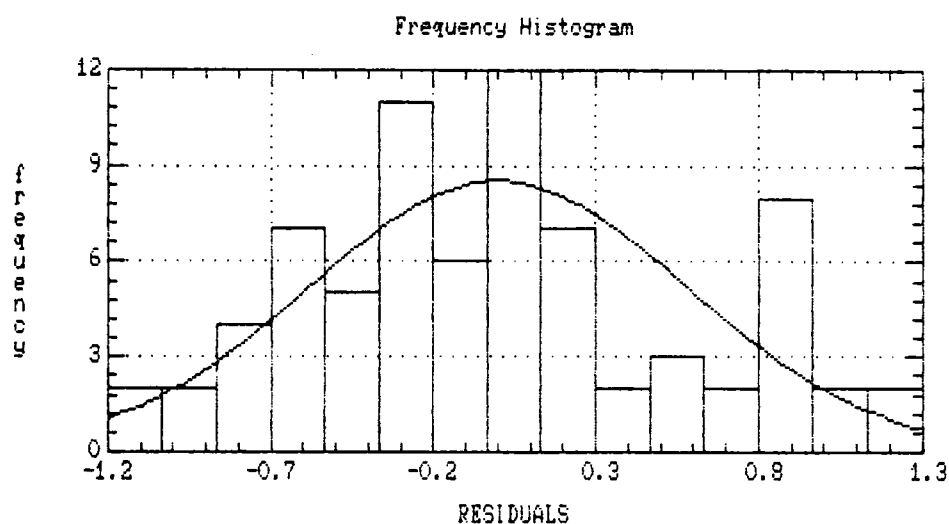
LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
15	20	-.157467	.014000	.145299	-0.2766	0.3046
30	20	-.090467	.081000	.145299	-0.2096	0.3716
60	20	.102533	.274000	.145299	-0.0166	0.5646
120	15	.193867	.365333	.167777	0.0298	0.7009

Press ENTER to continue.



LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
60	15	-.104133	.067333	.167777	-0.2682	0.4029
120	20	-.164467	.007000	.145299	-0.2836	0.2976
240	20	.174033	.345500	.145299	0.0549	0.6361
480	20	.068533	.240000	.145299	-0.0506	0.5306

Press ENTER to continue.



ESTIMATED PARAMETERS: 4.4779E-17 0.58511
 CHI*2 GOODNESS-OF-FIT STATISTIC = 10.675 WITH 6 DEGREES OF FREEDOM
 PROBABILITY OF A LARGER VALUE = 0.098951

***** STATISTICS *****

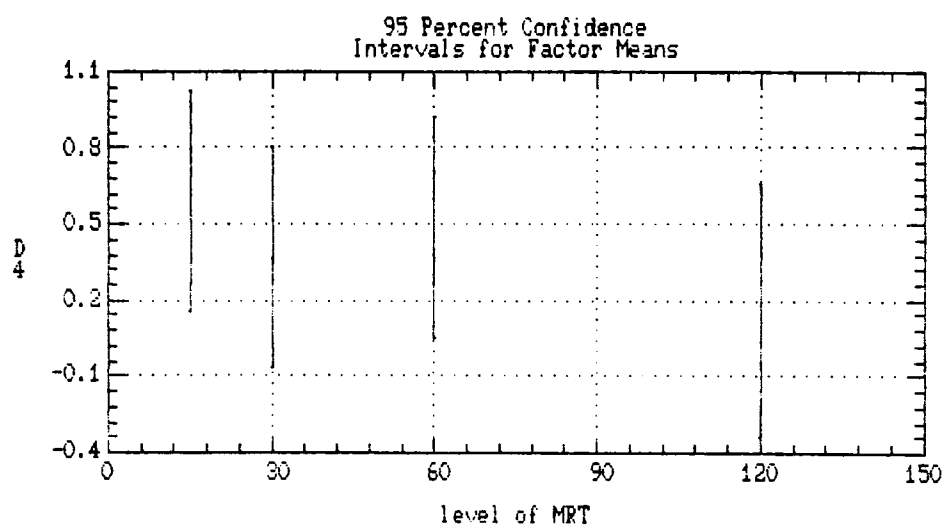
Onesample Analysis

RESIDUALS

Sample Statistics: Number of Obs.	75
Average	-1.1113E-1
Variance	0.34235
Std. Deviation	0.58511
Median	-0.03
Confidence Interval for Mean:	95 Percent
Sample 1	-0.13465 0.13465 74 D.F.
Confidence Interval for Variance:	0 Percent
Sample 1	
Hypothesis Test for H0: Mean = 0	Computed T statistic = -1.6449E-16
vs Alt: NE	Sig. Level = 1
at Alpha = 0.05	so do not reject H0.

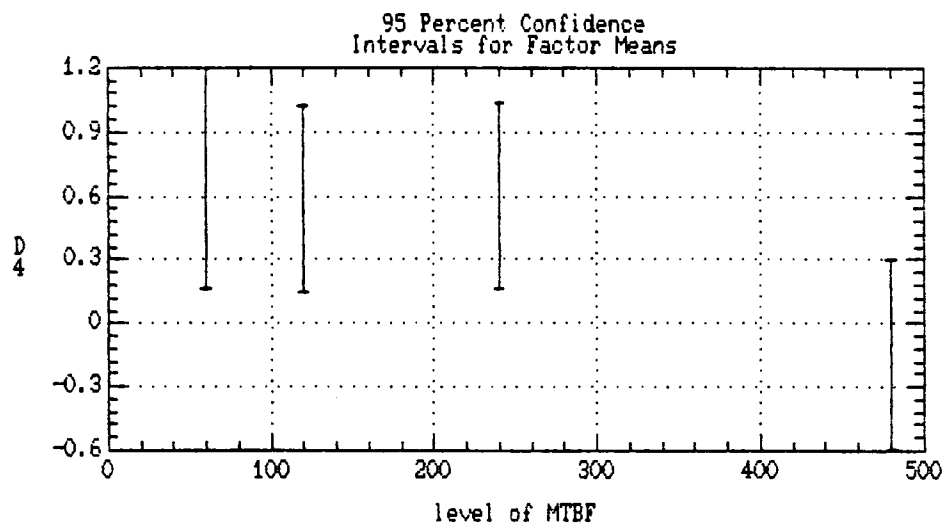
RESPONSE VARIABLE: D4					
SOURCE OF VARIATION	SUM OF SQUARES	D.F.	MEAN SQUARE	F-RATIO	PROB(>F)
MAIN EFFECTS	9.4692487	6	1.5782081	1.6800434	.1415
MRT	1.2771233	3	.4257078	.4531769	.7160
MTBF	7.7046183	3	2.5682061	2.7339219	.0515
2-FACTOR INTERACTIONS	8.2349700	8	1.0293713	1.0957923	.3788
MRT MTBF	8.2349700	8	1.0293713	1.0957923	.3788
RESIDUAL	56.363120	60	.9393853		
TOTAL (CORR.)	74.067339	74			

Press ENTER to continue.



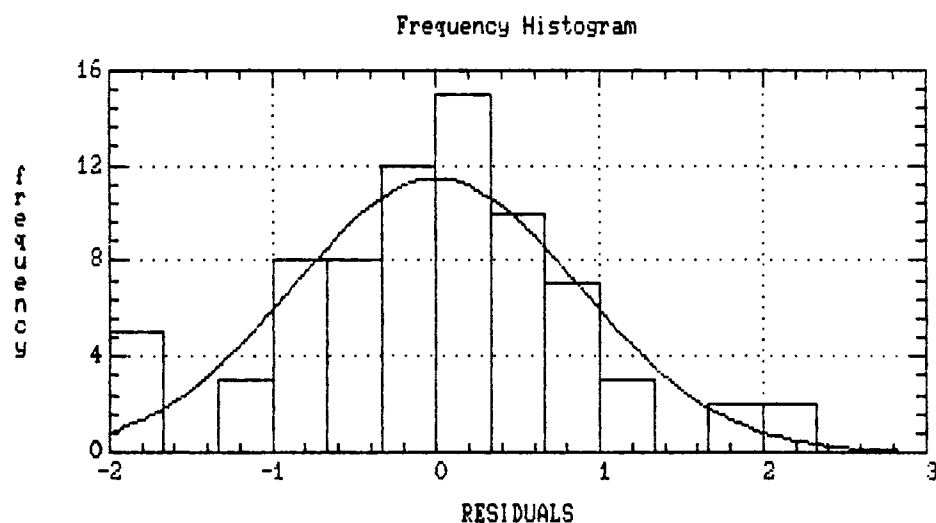
LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
15	20	.175867	.594000	.216724	-0.1606	1.0274
30	20	-.049633	.368500	.216724	-0.0649	0.8019
60	20	.067867	.486000	.216724	0.0526	0.9194
120	15	-.258800	.159333	.250251	-0.3412	0.6598

Press ENTER to continue.



LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
60	15	.251200	.669333	.250251	0.1688	1.1698
120	20	.174867	.593000	.216724	0.1596	1.0264
240	20	.182867	.601000	.216724	0.1676	1.0344
480	20	-.546133	-.128000	.216724	-0.5614	0.3054

Press ENTER to continue.



ESTIMATED PARAMETERS: 2.3685E-17 0.87273
 CHI² GOODNESS-OF-FIT STATISTIC = 2.4407 WITH 5 DEGREES OF FREEDOM
 PROBABILITY OF A LARGER VALUE = 0.7854

***** STATISTICS *****

Onesample Analysis

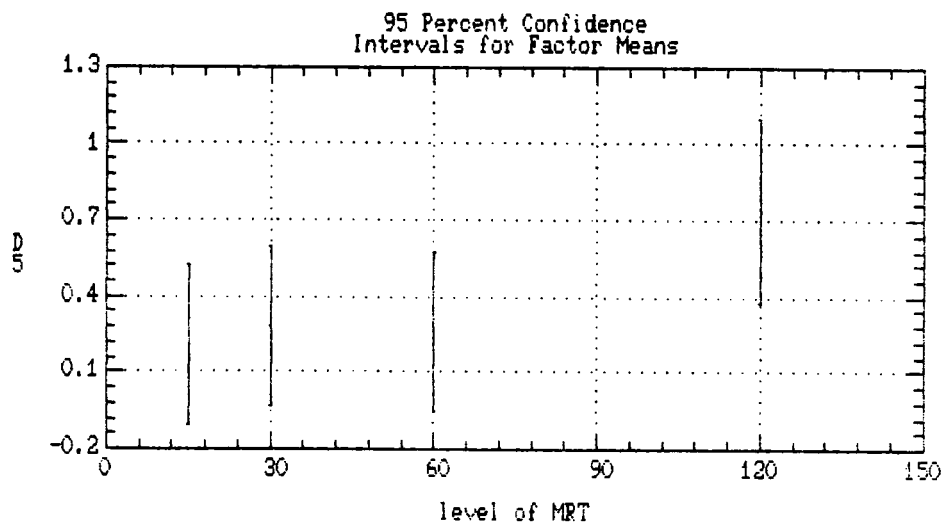
RESIDUALS

Sample Statistics: Number of Obs.	75
Average	2.5885E-17
Variance	0.76166
Std. Deviation	0.87273
Median	0.052
Confidence Interval for Mean:	95 Percent
Sample 1	-0.20084 0.20084 74 D.F.
Confidence Interval for Variance:	0 Percent
Sample 1	

Hypothesis Test for H ₀ : Mean = 0	Computed T statistic = 2.5686E-16
vs Alt: NE	Sig. Level = 1
at Alpha = 0.05	so do not reject H ₀ .

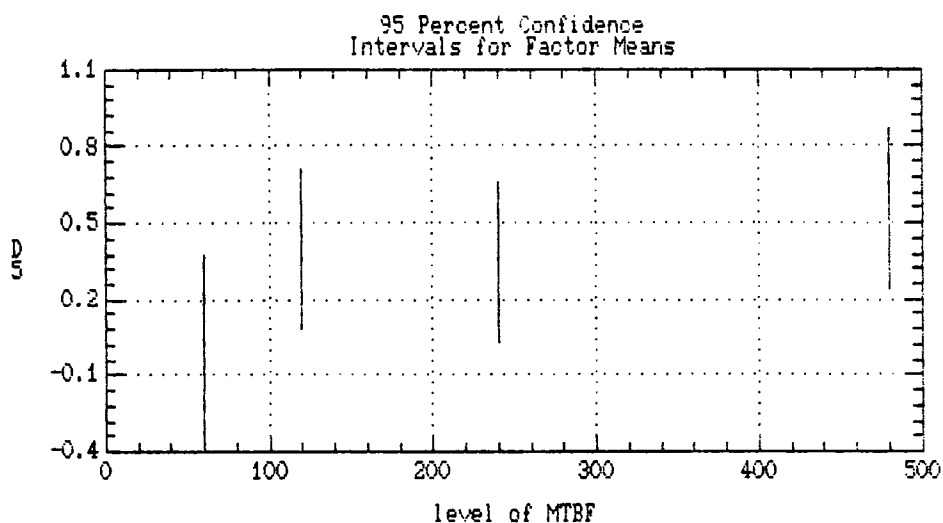
RESPONSE VARIABLE: D5					
SOURCE OF VARIATION	SUM OF SQUARES	D.F.	MEAN SQUARE	F-RATIO	PROB(>F)
MAIN EFFECTS	4.5398287	6	.7566381	1.5407070	.1806
MRT	1.9085933	3	.6361978	1.2954599	.2843
MTBF	1.6418100	3	.5472700	1.1143804	.3504
2-FACTOR INTERACTIONS	3.1198300	8	.3899788	.7940956	.6099
MRT MTBF	3.1198300	8	.3899788	.7940956	.6099
RESIDUAL	29.465880	60	.4910980		
TOTAL (CORR.)	37.125539	74			

Press ENTER to continue.



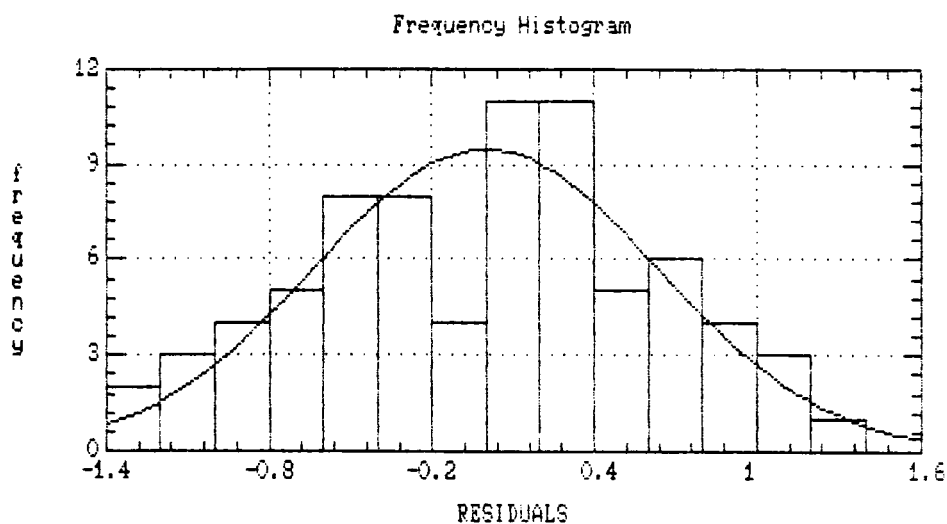
LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
15	20	-.136133	.212000	.156700	-0.1014	0.5254
30	20	-.069133	.279000	.156700	-0.0344	0.5924
60	20	-.087133	.261000	.156700	-0.0524	0.5744
120	15	.389867	.738000	.180942	0.3761	1.0999

Press ENTER to continue.



LEVEL	SAMPLE SIZE	EFFECT ESTIMATE	LEVEL AVERAGE	STANDARD ERROR	LOWER	UPPER
60	15	-.339467	.008667	.180942	-0.3532	0.3706
120	20	.048367	.396500	.156700	0.0831	0.7099
240	20	-.000633	.347500	.156700	0.0341	0.6609
480	20	.206867	.555000	.156700	0.2416	0.8684

Press ENTER to continue.



ESTIMATED PARAMETERS: -1.286E-17 0.63102
 CHI*2 GOODNESS-OF-FIT STATISTIC = 5.8662 WITH 7 DEGREES OF FREEDOM
 PROBABILITY OF A LARGER VALUE = 0.55545

***** S T A T G R A P H I C S *****

Onesample Analysis

RESIDUALS

Sample Statistics: Number of Obs.	75
Average	-1.4176E-1
Variance	0.39619
Std. Deviation	0.63102
Median	0.094
Confidence Interval for Mean:	95 Percent
Sample 1	-0.14522 0.14522 74 D.F.
Confidence Interval for Variance:	0 Percent
Sample 1	
Hypothesis Test for H0: Mean = 0	Computed T statistic = -1.9455E-16
vs Alt: NE	Sig. Level = 1
at Alpha = 0.05	so do not reject H0.

STATION UTILIZATION, STATION BUFFERS, RUN-TIME
AND TIME IN SYSTEM

TR: REPAIR TIME (min)						TBF: TIME BETWEEN FAILURES ALL STATIONS (min)						Station-Utilization					
Uniform						Exponential						Run					
												3					
												6					
												10					
												13					
												Run-Time					
												1					
												0.63					
												0.74					
												0.58					
												0.55					
												725.85					
												2					
												0.71					
												0.80					
												0.48					
												0.57					
												0.62					
												617.75					
												3					
												0.73					
												0.86					
												0.66					
												0.62					
												639.93					
												4					
												0.72					
												0.88					
												0.66					
												0.53					
												0.55					
												704.74					
												5					
												0.66					
												0.77					
												0.53					
												0.55					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					
												0.58					
												0.58					
												668.13					
												Average					
												0.69					
												0.81					

Run: 30-120

Station-Utilization					
Run	3	6	10	13	Run-Time
1	0.58	0.58	0.31	0.29	978.33
2	0.58	0.66	0.50	0.38	978.00
3	0.46	0.66	0.37	0.28	1107.29
4	0.51	0.56	0.41	0.29	1150.36
5	0.56	0.62	0.50	0.38	977.22
Average	0.54	0.61	0.42	0.32	1038.24

In-Buffer					
Run	3	6	10	13	Tsys
1	0.12	0.25	0.28	0.40	93.57
2	0.04	0.25	0.12	0.09	79.58
3	0.03	0.29	0.09	0.14	82.87
4	0.02	0.17	0.15	0.34	88.21
5	0.02	0.17	0.11	0.10	71.49
Average	0.05	0.22	0.15	0.21	83.14

Run: 60-60

Station-Utilization					
Run	3	6	10	13	Run-Time
1	0.12	0.10	0.09	0.07	7748.01
2	0.20	0.22	0.11	0.10	4488.39
3	0.21	0.27	0.12	0.13	3620.98
4	0.17	0.20	0.11	0.08	4994.83
5	0.16	0.14	0.10	0.08	5403.69
Average	0.17	0.19	0.11	0.09	5251.18

In-Buffer					
Run	3	6	10	13	Tsys
1	0.00	0.09	0.06	0.05	157.74
2	0.00	0.05	0.15	0.06	124.91
3	0.01	0.12	0.19	0.18	155.90
4	0.00	0.10	0.13	0.11	148.47
5	0.01	0.11	0.13	0.17	165.55
Average	0.00	0.09	0.13	0.11	150.51

Run: 30-240

Station-Utilization					
Run	3	6	10	13	Run-Time
1	0.64	0.54	0.44	0.45	744.00
2	0.63	0.70	0.56	0.47	750.60
3	0.65	0.80	0.43	0.60	690.25
4	0.61	0.66	0.45	0.43	891.52
5	0.67	0.76	0.50	0.49	786.05
Average	0.64	0.69	0.48	0.49	772.48

In-Buffer					
Run	3	6	10	13	Tsys
1	0.01	0.43	0.41	0.12	78.55
2	0.01	0.29	0.08	0.31	72.02
3	0.01	0.47	0.33	0.41	82.37
4	0.07	0.25	0.04	0.08	72.48
5	0.01	0.28	0.09	0.09	72.20
Average	0.02	0.34	0.19	0.20	75.52

Run: 60-120

Station-Utilization					
Run	3	6	10	13	Run-Time
1	0.28	0.32	0.22	0.16	2218.10
2	0.41	0.49	0.39	0.29	1431.64
3	0.25	0.34	0.19	0.17	2784.63
4	0.23	0.24	0.17	0.18	2628.70
5	0.26	0.28	0.20	0.14	2736.92
Average	0.28	0.33	0.23	0.19	2360.00

In-Buffer					
Run	3	6	10	13	Tsys
1	0.01	0.08	0.32	0.26	113.41
2	0.01	0.09	0.45	0.34	109.50
3	0.00	0.07	0.07	0.19	103.93
4	0.02	0.12	0.15	0.22	107.69
5	0.02	0.18	0.18	0.12	129.35
Average	0.01	0.11	0.23	0.22	112.77

Run: 30-480

Station-Utilization					
Run	3	6	10	13	Run-Time
1	0.64	0.72	0.57	0.42	817.10
2	0.70	0.90	0.68	0.57	640.23
3	0.68	0.87	0.57	0.63	656.39
4	0.72	0.83	0.53	0.53	626.20
5	0.67	0.82	0.64	0.51	675.60
Average	0.68	0.83	0.60	0.53	683.10

In-Buffer					
Run	3	6	10	13	Tsys
1	0.01	0.20	0.31	0.17	79.87
2	0.02	0.90	0.28	0.09	81.36
3	0.03	0.35	0.28	0.48	80.64
4	0.02	0.26	0.37	0.21	69.80
5	0.01	0.37	0.30	0.13	72.38
Average	0.02	0.42	0.31	0.21	76.81

Run: 60-240

Station-Utilization					
Run	3	6	10	13	Run-Time
1	0.45	0.54	0.36	0.31	1278.00
2	0.48	0.58	0.46	0.46	1040.20
3	0.54	0.72	0.51	0.43	828.00
4	0.45	0.50	0.36	0.32	1230.85
5	0.50	0.58	0.48	0.51	1019.50
Average	0.49	0.58	0.43	0.40	1079.31

In-Buffer					
Run	3	6	10	13	Tsys
1	0.01	0.15	0.11	0.35	88.72
2	0.02	0.33	0.37	0.13	93.92
3	0.01	0.37	0.44	0.32	89.77
4	0.01	0.27	0.01	0.05	71.02
5	0.14	0.36	0.43	0.71	119.07
Average	0.04	0.30	0.27	0.31	92.50

Run: 60-480

Station-Utilization Run	3	6	10	13	Run-Time
1	0.77	0.74	0.56	0.44	712.90
2	0.52	0.72	0.59	0.41	919.50
3	0.63	0.76	0.58	0.48	747.60
4	0.59	0.75	0.53	0.34	813.60
5	0.68	0.79	0.71	0.54	663.75
Average	0.64	0.75	0.59	0.44	771.47

In-Buffer Run	3	6	10	13	Tsys
1	0.04	0.44	0.13	0.50	80.88
2	0.01	0.45	0.68	0.11	100.00
3	0.10	0.52	0.25	0.06	77.62
4	0.01	0.59	0.20	0.73	93.55
5	0.02	0.23	0.37	0.02	68.96
Average	0.03	0.45	0.33	0.29	84.20

Run: 120-120

Station-Utilization Run	3	6	10	13	Run-Time
1	0.16	0.18	0.17	0.13	4758.00
2	0.38	0.30	0.33	0.21	2364.79
3	0.17	0.17	0.08	0.08	5642.03
4	0.20	0.19	0.15	0.12	4671.37
5	0.24	0.42	0.21	0.17	2541.64
Average	0.23	0.25	0.19	0.14	3995.57

In-Buffer Run	3	6	10	13	Tsys
1	0.03	0.04	0.19	0.16	141.52
2	0.00	0.14	0.62	0.59	175.89
3	0.00	0.09	0.21	0.16	175.41
4	0.00	0.11	0.20	0.15	157.84
5	0.01	0.29	0.11	0.60	153.77
Average	0.01	0.13	0.26	0.33	160.89

Run: 120-240

Station-Utilization Run	3	6	10	13	Run-Time
1	0.35	0.52	0.32	0.37	1397.65
2	0.38	0.25	0.26	0.18	2313.00
3	0.28	0.35	0.23	0.16	2223.05
4	0.29	0.30	0.24	0.18	2575.32
5	0.36	0.34	0.32	0.26	1803.00
Average	0.33	0.35	0.27	0.23	2062.40

In-Buffer Run	3	6	10	13	Tsys
1	0.01	0.08	0.35	0.72	114.47
2	0.07	0.21	0.11	0.38	137.01
3	0.00	0.28	0.34	0.21	131.87
4	0.05	0.23	0.14	0.17	128.33
5	0.00	0.07	0.35	0.49	125.35
Average	0.03	0.18	0.26	0.39	127.41

Run: 120-480

Station-Utilization Run	3	6	10	13	Run-Time
1	0.75	0.72	0.56	0.50	721.55
2	0.49	0.58	0.44	0.55	910.35
3	0.53	0.58	0.48	0.41	1083.00
4	0.47	0.57	0.34	0.35	1188.05
5	0.47	0.49	0.38	0.38	1185.76
Average	0.54	0.59	0.44	0.44	1017.74

In-Buffer Run	3	6	10	13	Tsys
1	0.02	0.36	0.12	0.07	69.61
2	0.01	0.32	0.08	1.05	96.38
3	0.01	0.29	0.36	0.25	96.07
4	0.01	0.29	0.10	1.15	117.49
5	0.02	0.19	0.24	0.87	115.79
Average	0.01	0.29	0.18	0.68	99.07

APPENDIX H

DERIVATION OF FORMULAS

In Chapter 6, section 6.4, we needed to find the average time a station i was operational up to a time τ . In this appendix we show a derivation for the case when the time between failures (T_f) and the repair time (T_r) have exponential distributions. Under this assumption we can apply basic concepts of birth-and-death processes.

At any time t , a machine is in one of two states:

State 1: machine is working and available.

State 0: machine is failed and being repaired.

Let θ be the failure rate and μ the repair rate (here we assume them constant but the procedure below also applies when both rates vary in time).

$$P_1(t+h) = P_0 \mu h + P_1 (1-\theta h) + o(h) \quad (H-1)$$

$$P_0(t+h) = P_0 (1-\mu h) + P_1 \theta h + o(h)$$

As h goes to zero,

$$P_1'(t) = \mu P_0 - \theta P_1 \quad (H-2)$$

$$P_0'(t) = -\mu P_0 + \theta P_1$$

In the steady state, the stationary probabilities π do not depend on time and furthermore $\pi_0 + \pi_1 = 1$. Then,

$$\pi_1 = 1/(1 + \theta/\mu) \quad (\text{H-3})$$

$$\pi_0 = 1/(1 + \mu/\theta)$$

This means that the probability that the machine is available is π_1 . The expression for π_1 can be easily recognized as:

$$\pi_1 = 1/(1 + mtr/mtbf) \quad (\text{H-4})$$

or,

$$\pi_1 = 1/r \text{ as defined in section 6.4.}$$

Thus, given any interval of length τ , the expected time in which the machine is available can be taken as:

$$\text{Expected Time Available} = \tau/r \quad (\text{QED}).$$

By solving (H-2), it is easy to show also that:

$$\begin{aligned} P_1(t) &= \pi_1 \left(1 + \theta/\mu e^{-(\theta+\mu)t} \right) \\ P_0(t) &= \pi_0 \left(1 - e^{-(\theta+\mu)t} \right) \end{aligned} \quad (\text{H-5})$$

VITA

Juan Miguel Sepúlveda Salas was born in Santiago, Chile, on May 7, 1958. In 1975, he enrolled the University of Santiago of Chile (USACH) and graduated in 1981 with a degree in Industrial and Systems Engineering.

Between 1980 and 1981, Mr. Sepúlveda taught programming languages at DUOC, a Catholic University's professional institute in Santiago, and worked as a part-time systems analyst and minicomputer applications developer for small commercial firms.

Upon graduation, he accepted a full-time position at the Industrial Engineering Department at USACH, where he taught Operations Research, Simulation, and Statistical Inference, for IE and Computer Science majors.

In Fall 1982, USACH granted him a scholarship and he took leave of absence to enroll The University of Tennessee at Knoxville.

He obtained the Master of Science degree in Industrial Engineering in 1985 and continued his work towards the PhD degree.

Mr. Sepúlveda will join the Industrial Engineering Department at USACH after graduation.