A Real-Time Dispatch Policy for a System Subject to Sequence-Dependent, Random Setup Times

by

Mitchell H. Burman

B.A. in Mathematical Economics, Colgate University, USA (1985)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1993

Author .....
Department of Electrical Engineering and Computer Science
November 31, 1992

Certified by .....
Stanley B. Gershwin
Senior Research Scientist
Thesis Supervisor

Accepted by .....
Richard Larson
Co-Director, Operations Research Center
Chairman, Interdepartmental Committee on Operations Research

A Real-Time Dispatch Policy for a System Subject to Sequence-Dependent, Random Setup Times

by

Mitchell H. Burman

Submitted to the Department of Electrical Engineering and Computer Science
on November 31, 1992, in partial fulfillment of the
requirements for the degree of
Master of Science in Operations Research

Abstract

A closed-loop dispatch policy for a medical device manufacturing facility is proposed. This facility is subject to strict due dates and significant, sequence-dependent, random setup times. Planners at the plant presently rely on techniques which only account for first moment characteristics of activity durations. Based on informal observations, experienced machine operators reacting intuitively in real time to random events can outperform planners' schedules.

An order for a number of part types, all with the same fixed due date, is provided once every week to the factory along with projections for future demand. A two-level method is proposed which first sequences lots and then sizes lots dynamically in response to system events. The lot-sequencing algorithm has the advantages that it is quick, needs to be solved only once per week and limits the amount of data which must be maintained. A sample lot-sizing problem is proposed and solved numerically. From the characterization of the solution, a closed-loop, threshold, control heuristic is developed. While setup times are the only random activity accounted for in this model, we believe the methodology can be extended to incorporate many phenomena, and that closed-loop feedback control may be a practical way for viewing many dispatch level scheduling problems. A lot-sizing implementation is discussed.

Thesis Supervisor: Stanley B. Gershwin
Title: Senior Research Scientist

Acknowledgments

This thesis is dedicated to my father, whose advice to never be afraid to fail, to try and to be honest has remained my most important asset; to my mom for her emotional support and for her reminders to have fun; and to my Brother Steven, Sister Karen and Uncle George for their faith in my ability.

Thanks to Dr. Stanley B. Gershwin (Stan) for his emotional support and guidance on my thesis; to Richard Larson (Dick) for daring me to be different; and to John Matson, George Thompson, Goerge Brandenburg, Jay Silverstein, Bob Delia, and Don Nociolo. Special thanks to Mark Duggan for programming the software prototype.

The author also wishes to acknowledge the Leaders for Manufacturing Program for its support of this work.

Contents

1 Introduction **7**

1.1 Background and Objectives 8

1.2 Literature Review 9

1.3 Paper Summary 11

2 Factory Description **13**

2.1 General Definitions 13

2.2 Physical Description 15

2.3 Important Phenomena 19

2.4 Scheduling Objectives 22

2.5 Current Approaches to the Problem 24

3 Solution Approach **27**

3.1 Methodology Descriptions 28

3.2 Model Premises 31

3.2.1 Isolated Bottleneck 31

3.2.2 Reasonable Demands by Management 32

3.2.3 Separation of Lot Sequencing and Lot Sizing 32

3.2.4 Efficiency of Frequent Near-Optimal Control 33

3.3 System Hierarchy 33

3.4 Part Dispatching Methodology 38

3.4.1 Dispatching Hierarchy – Two-Level Method 39
3.4.2 Setup Sequencing – The $\mathcal{L}$ *List* 40
3.4.3 Dynamic Lot Sizing 41

4 Setup Hierarchy **43**
4.1 Sequencing Literature Review 45
4.2 Notation and Definitions 47
4.3 The Hierarchical Setup Structure 49
4.3.1 Setup Structure 49
4.3.2 Hierarchy Definition 51
4.4 Traveling Salesman-Type Formulation 52
4.4.1 Assumptions 52
4.4.2 General Formulation – Setup Space 53
4.4.3 Hierarchical Formulation 54
4.5 The Algorithm 56
4.5.1 Optimality Proof 56
4.5.2 Bounds 60
4.5.3 Formal Algorithm 63
4.6 Algorithm Attributes 64
4.7 Summary 66

5 Real-Time Lot-Sizing Policy **68**
5.1 Lot-Sizing Literature 69
5.2 Problem Description 70
5.3 The Dynamic Program 73
5.3.1 Solution Technique – Dynamic Programming 73
5.3.2 The Model 74
5.4 An Example 79
5.4.1 Numerical Derivation of Control Policy 79

5.4.2 Other Thresholds 82
5.4.3 The Optimal Threshold and the Heuristic 86
5.5 Summary 86

6 Simulation Experiments **87**
6.1 Simulation Test Design 88
6.2 Open-Loop Policies Used in Practice 88
6.3 Closed-Loop Controllers 91
6.4 Test Results 94
6.5 Summary 95

7 Extensions and Implementation **99**
7.1 Project Testbed 100
7.2 Theoretical Extensions 101
7.2.1 Lot-Sequencing Extensions 101
7.2.2 Lot-Sizing Extensions 102
7.3 The Software 106
7.4 Future Extensions 109

8 Summary and Conclusion **111**
8.1 Summary 111
8.2 Future Work 113
8.3 Conclusion 113

List of Figures

2-1 Production Flow 15

3-1 Symbolic Hierarchy 29

3-2 A Feedback Loop 30

3-3 Management Structure 34

3-4 Spatial Decomposition 35

3-5 Temporal Decomposition 37

3-6 Two-Level Method 39

4-1 Part Type Sequencing Algorithm Flow 43

4-2 Setup Hierarchy 51

4-3 Graphical Representation of Minimum Setup Time Problem 53

4-4 Hierarchical Setup Space 55

4-5 Two-Level Hierarchical System 57

5-1 Decision Points. 75

5-2 Control Policy 80

5-3 Other Thresholds 83

6-1 Closed-Loop Controller Thresholds 92

6-2 Expected Number of Extra Jobs Completed 97

6-3 Percentage of Weeks with Incomplete Current Requirements 98

7-1 Excess Time Allocation Decision 104
7-2 Allocation of Discretionary Activities From $\mathcal{P}$ *Set* 108
7-3 High Level Feedback Loop 109

List of Tables

3.1 System Events and Their Frequencies 36

4.1 Machine Configurations 48

4.2 Setup Parameters 50

5.1 Example Parameters 79

6.1 Performance Summary 96

Chapter 1

Introduction

In this thesis, we present a real-time, closed-loop method of scheduling a system bottleneck subject to significant, sequence-dependent, random setup times. Demands for a number of part types, all with the same fixed due date, are provided every week along with limited future demand projections to a manufacturing facility. In contrast to more traditional methods which first set lot sizes and then sequences them, our scheduling technique first establishes a part type sequence which is then used as an input for a real-time, lot-sizing algorithm. This problem was motivated by a medical device manufacturing facility.

Currently, neither practitioners nor theorists provide an adequate solution to our problem. Planners responsible for scheduling at the plant presently rely on techniques which only account for first moment characteristics of setup, failure and operation durations. Based on informal observations, experienced floor managers who react intuitively in real time to random disruptive events can outperform planners' schedules. Scheduling theorists frequently model manufacturing systems using unrealistic assumptions to obtain tractable analytical solutions. In addition,

these models require significant time and effort to solve, preventing their use in a real-time environment. One typical modeling simplification involves assuming that stochastic events may be treated deterministically by assuming they occur in accordance with their expected behavior. Observations of many manufacturing facilities has led to the conclusion that any scheduling philosophy must account for the variability associated with random demand, machine failures, significant setup times and inconsistent processing times.

Our work began by identifying an industry problem from which this research could develop. A description of this effort is outlined (Section 1.1). Second, a literature review is provided (Section 1.2). Finally, a paper summary is presented (Section 1.3).

1.1 Background and Objectives

This research began with a three-month assignment at a manufacturing plant that produces medical devices used by hospitals and medical facilities. The objective established by management was to design a real-time scheduling methodology for the facility. There were no restrictions.

During the first few weeks, the primary goal was to gain as good an understanding of the process as possible. This involved talking with the machine operators for their perspective on the important scheduling issues. A number of issues could be clearly identified as the most important and difficult. The process contained one stage which was significantly more expensive than all the other stages. The machines at this stage were subject to failures and required setups to make over 700 different part types. These setups were also significant, random and sequence-dependent. Each machine would potentially need to produce as many as 40 different parts within one week.

The next step was to analyze the system, study the existing literature and examine the currently available scheduling software. The system conditions always violated one or more basic assumption in the previous research. Almost none of the existing software was designed to accommodate real-time control in response to system events. The software was also extremely expensive and required highly trained operators. Something new had to be developed to meet management's objective. The final weeks of the assignment were spent outlining the system objectives in more detail in preparation for developing a scheduling procedure which would be implementable and theoretically sound. The method had to be understood by both upper management and production planners.

A two-level dispatch policy was suggested and analyzed. At the higher level, part types are sequenced. This sequence is then used as input to a real-time lot-sequencing (lower level). This thesis provides the theoretical foundation of the policy and discusses the development of a rudimentary prototype controller. This prototype handles only 7 part types as opposed to 700. For this case, finding the minimum total setup time lot sequence could easily be handled with an exhaustive search. This allowed the prototype experiment to focus strictly on determining lot sizes (lower level) in real time. It also allowed us to test the algorithm's ability to accommodate additional variability other than setup times.

1.2 Literature Review

The current manufacturing paradigm is switching from the old view of a deterministic system waiting to be optimized (Taylor, 1947) to a new dynamic system in which the rules constantly change (Hayes, Wheelwright and Clark, 1988). Demand is random, machines break down, setups take time, etc. It is not that these issues never existed, but rather production systems were not being pushed to the

limits where these issues potentially forced the system into an unstable state (Neeley, 1990). The reason for this instability can be easily demonstrated in queueing theory. When the arrival rate approaches the service rate, the probability of the queue growing to an infinite length is 1 (Kleinrock, 1975).

The goal of this thesis is to develop a policy which is based on a real problem and which can be implemented. Recent literature reviews of production research identify practical research as a domain which has been neglected (Graves, 1981; Rodammer and White, 1988). Burman (1992) concludes that very little academic research on stochastic control is being applied to production systems. This is due, at least in part, to the disparity between the complexity of actual factories and the limiting assumptions of most algorithms. Most algorithms treat systems as if they are deterministic.

Algorithms are required which account for disruptive events in real time (Gershwin, 1989; Violette and Gershwin, 1991). However, efforts to optimize systems at the dispatching level frequently lead to the well known curse of dimensionality due to the vast number of events occurring in most manufacturing operations. Work in Gershwin (1993) reduces the complexity of the problem by treating the rates of phenomena rather than discrete events, while employing inherently stable control approaches. The method advocates beginning each system analysis by decomposing the overall scheduling problem. *Hierarchical decomposition* breaks down long-term, overall system goals into more refined, short-term, subsystem goals which are more easily modeled, analyzed and controlled. (Burch, Oliff and Sumichrast, 1987; Gershwin, 1987, 1989; Gershwin, Caramanis, and Murray, 1988).

This decomposition is used in conjunction with a *hierarchical flow control* method which combines rate approximations and feedback to compensate for randomness. Activities and events are classified by their degree of controllability. *Controllable events* include production operations, setup changes, preventative maintenance; machine failures and setup completion times are *uncontrollable events.*

D'Angelo, Caramanis, Finger, Mavretic, Phillis and Ramsden (1988) look at event driven models of production lines in the presence of uncontrollable events. We also employ feedback control to dynamically schedule the system in response to uncontrollable events.

This concept of a hierarchy for low level production planning is not new (Bitran and Hax, 1977). However, the traditional precedent has been to let lot sizing dominate lot sequencing and to treat the system as if it is deterministic. For example, casual observations of traditional planning operations indicate that corporate management establish lot sizes (detailed *economic lot sizing problem* literature review in Section 5.2) and due dates at a high level, leaving the task of lot sequencing (detailed setup and sequencing literature review in Section 4.2) as the responsibility of the factory operators. We separate the two functions of lot sequencing (Level 1) and lot sizing (Level 2) into two different levels with lot sequencing dominating (Figure 3-6). Based on this hierarchical structure we develop a new paradigm for scheduling systems with significant, sequence-dependent, random setup times. This method suggests using greedy algorithms to establish part type sequences as an input to a real-time lot-sizing algorithm.

1.3 Paper Summary

A description of the factory is provided in Chapter 2. Chapter 3 outlines the overall approach to scheduling the system which establishes a two-level approach of lot sequencing and lot sizing. A setup hierarchy structure is explored and exploited in establishing a greedy algorithm for the sequencing of setups in Chapter 4. This method is not only fast, but has the advantage of drastically reducing the amount of data required for scheduling. Chapter 5 establishes a dynamic programming formulation from which a lot-sizing control policy is developed. Numerical

tests are conducted which compare the derived control policy with other current manufacturing control practices in Chapter 6. Chapter 7 describes extensions to the basic theory incorporated in a prototype controller. Finally, a list of future research directions and concluding remarks are provided in Chapter 8.

Chapter 2

Factory Description

In this chapter, the manufacturing facility introduced in Chapter 1 is described. A list of definitions is provided (Section 2.1). The physical attributes of the system are detailed (Section 2.2). An outline of the important phenomena related to scheduling (Section 2.3) and management's objectives (Section 2.4) are presented. Finally, currently employed scheduling practices are discussed (Section 2.5).

2.1 General Definitions

The definitions of the concepts of manufacturing, manufacturing systems, lots, setups and capacity are fuzzy, overlap with each other and continually change. In addition, these definitions may vary with one's perspective of a system. For example, a *production process* is the full sequence of material transformations required to manufacture a part. The factory manager may view this as a sequence of machine operations, while the machine operator may view it as one cycle on

an injection molding machine. Issues relating to perspective plague the literature. Therefore, precise definitions of these concepts are provided for this thesis.

Manufacturing is the transformation of material into something useful and portable (Gershwin, 1993). *Raw materials* are the input in manufacturing and *parts* are the output. *Process steps* or *stages* are smaller subdivisions of the overall process usually concentrating on the material transformations that occur at a particular machine. *Production planning* consists of simultaneously meeting demand and controlling material flows, subject to system constraints such as capacity. A *manufacturing system* facilitates the production plan and consists of a set of machines, transporting elements, computers and storage buffers.

Manufacturing environments are distinguished from one another in terms of how they change over time, how material flows through them, the repeatability of the process, routing restrictions and flexibility. *Dynamic systems* change with time, *static systems* do not, *stochastic systems* change randomly and *deterministic systems* may change or remain static but without any associated randomness. The concepts developed in this thesis focus on high volume, discrete part (cars, computers, lightbulbs, etc.) as opposed to job shop (machine shop, custom cabinets, etc.) or continuous (chemicals, petroleum, etc.) production facilities. Critical production planning issues in these environments include loading and sequencing policies, buffer management and demand fulfillment.

An *elementary lot* is the smallest number of parts that a process is capable of producing at once. A *lot* is a group of consecutively processed elementary lots of the same part type. A *batch* is similar to a lot, except that the elementary lots are not required to be of the same part type. The *machine configuration* is the physical state of the machine, and it determines the kind of parts which can be manufactured at any point in time. A *setup* is a change in the machine configuration. The *setup time* is the time required to switch machine configurations between the production of two different part types. *Random setups* are setups whose setup

times are not deterministic. *Sequence dependent* setup times are functions of the pre- and/or post-setup machine configurations.

Capacity henceforth refers strictly to a system's capability to produce, subject to a number of system constraints (*e.g.*, return on investment, work in process, lead time, quality and required setup frequencies). In the facility discussed in this thesis, the system is a single machine and capacity is measured by the available time for producing parts on the machine.

2.2 Physical Description

In this section, a description of the manufacturing process (Figure 2-1) is provided. The first stage of this process is the focus of this research (Figure 2-1, Stage One Bottleneck). The system downstream of Stage One is outlined briefly only to highlight the reasons why it is ignored in the algorithms eventually developed.

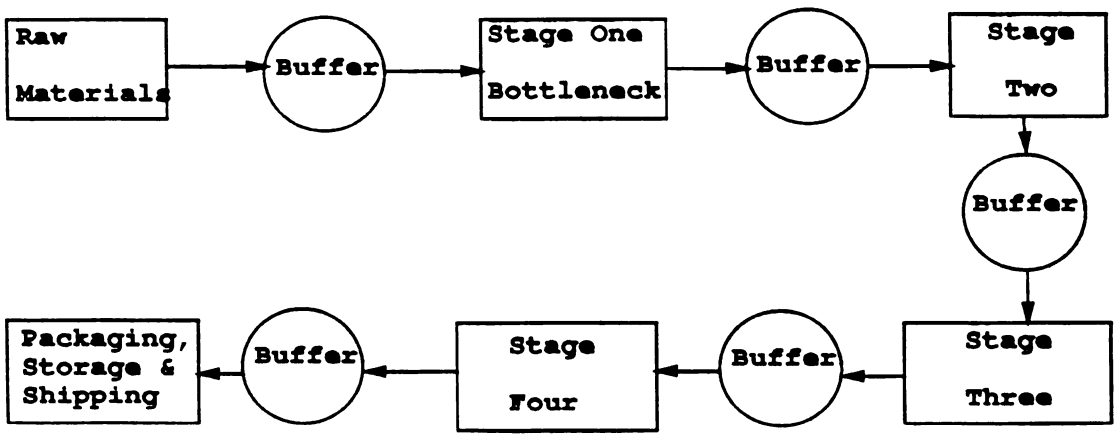

Figure 2-1: Production Flow

Raw Materials

Six different grades of raw material frequently arrive to the factory. The material is inexpensive, requires little space and must go through a pre-processing step prior to entering Stage One. Without pre-processing, Stage One machines are much more likely to fail due to damage from the unrefined material. These pre-processing machines are inexpensive compared to other machines in the system and also require very little space. Pre-processing setup times are minimal, allowing frequent setups and affording maximum system flexibility at a minimum cost in system capacity. Based on these characteristics, it is fair to assume that there is an unlimited supply of raw material.

Stage One

Stage One consists of a set parallel refining machines, each capable of transforming raw materials into elementary lots of many different part types. Processing times for all part types are identical. Each Stage One machine costs an order of magnitude more than any other stage in the process and requires significant space and manpower to operate.

Labor and tooling are both fixed costs associated with setup changes. Setup time is the only variable cost. Stage One setup times are significant, sequence dependent (Section 2.3) and random. Setup time is a major determinant of machine capacity utilization (Section 2.4) and must be carefully planned. Scheduling frequent setup changeovers such as advocated in Just-in-Time (JIT, Golhar and Stamm, 1991) and Continuous Flow Manufacturing (Gooch, George and Montgomery, 1987) may result in incomplete orders due to the machine time consumed during setups.

While labor availability is not a constraint on the machine's ability to change

setups, available tooling does affect the scheduling of the system. Redundant tooling would allow quick setup changes and would decrease setup times. Unfortunately, Stage One tooling is very expensive (in some cases, more expensive than the machines themselves). Each part type requires slightly different tooling, making excess tooling an infeasible way of reducing setup times. The significant tooling cost also imposes a second constraint on the system. Without excess tooling, each part type may only be produced on a specific machine. This reduces system flexibility (Section 2.4). Thus, part type routing through the first stage is deterministic. That is, each of the part types is assigned to one and only one machine.

Process documentation and quality testing consume a significant portion of setup time. Strict federal guidelines associated with medical products require lengthy documentation and safety inspections which cannot be reduced in the short term. Therefore, while reducing setup times is a good system goal, significant setup times cannot be eliminated in the near future and must be accounted for in the scheduling methodology.

Downstream System

The high cost of Stage One machines suggests that sufficient downstream capacity be designed into the process so that Stage One is never blocked. This extra capacity is available. The combination of the unlimited raw materials and this excess downstream capacity creates a long and short term system bottleneck at Stage One (Goldratt and Cox, 1986). Since this thesis eventually concentrates only on the system bottleneck, only a cursory outline of the later stages is given. In addition, inventory and buffers are mentioned only briefly because of their minor role in the suggested method for scheduling the facility.

Stage Two A *batch process* transforms a large number of elementary lots at the same time. Usually, the process is not started until a certain number of lots are available for processing. Once the critical number is reached, all the lots are processed together and completed at the same time. An example of a batch process is an oven in which all parts are loaded and the oven door is sealed. No part may enter or leave the process while the door is sealed. Upon completing Stage One, part types are grouped into batches for a second process (Figure 2-1, Stage Two). No setup time is required for processing different part types, and multiple part types may be batched together. The processing times at this stage are significant.

Stage Three A *disassembly process* separates elementary lots into individual parts. After completing Stage Two, the elementary lots are disassembled (Figure 2-1, Stage Three). These disassembly machines are inexpensive and have virtually no setup time. This allows great routing flexibility. However, utilization of these machines is erratic because with the completion of every batch at Stage Two, a surge of parts flows to disassembly. To foster a smooth flow through this process, a great deal of capacity is necessary. However, the processing at this stage are small enough so as not to cause any delay.

Stage Four A *continuous flow process* is the opposite of a batch process. Parts traveling through the system may be viewed as flow which may be turned on and off arbitrarily. Discrete part production may also be viewed as a continuous flow process when the elementary lot size is 1 or close to it. The small lot size allows the system to be closely approximated by a flow. After Stage Three, the individual parts proceed to a continuous flow process (Figure 2-1, Stage Four) where they are refined into the final product. The processing time at this stage similarly cause no delay.

Final Stages Prior to packaging the finished product for shipping, the product is tested, analyzed and counted. Packing is the last stage in the process, after which finished assemblies are stored in a finished goods inventory. There are no significant issues requiring discussion at these stages.

Buffer and Storage Space

A *buffer* is a physical space or piece of equipment designated for storing material between process stages. *Inventory* is the partially processed material stored in the buffer. Inventory has very low cost, little obsolescence, and requires little space. *Blockage* is the interruption of production on a machine due to the buffer immediately *downstream* (later in the material flow) being full of inventory, not allowing parts to flow from the machine. *Starvation* is the interruption of production on a machine due to the buffer immediately *upstream* (earlier in the material flow) being empty of inventory, starving the machine of its raw material. The placement of buffers and the policies governing inventory may have a great effect on system performance because of their effect on blockage and starvation. However, we assume that sufficient inventory, ample raw material, excess downstream capacity and buffer space exist so that these are not issues in the model of Stage One (Section 3.2).

2.3 Important Phenomena

There are a number of important phenomena affecting production planning at the facility including setup variability, variance of demand, machine availability and the influence of management decisions.

Setup Variability and Sequence Dependence

In addition to the significant amount of setup time each week at Stage One, setup time variability is also important. This exists for a number of reasons including machine calibration times and process documentation requirements (*i.e.*, reports on how much time the setup took, sample assemblies for quality control, etc.). Both of these tasks take variable amounts of time. Therefore, this randomness must be accounted for in the production planning methodology.

A second important issue is setup sequence dependence (Section 2.1). For example, producing Part Type 1 then Part Type 2 then Part Type 3 may take a significantly different amount of time than producing Part Type 3 then Part Type 2 and then Part Type 1. Therefore, the sequence of setup changes is a very important factor in reducing the overall setup time. Chapter 4 focuses on dealing with the sequencing of setups to minimize expected total setup time.

Demand Variability

Demand requirements arrive every Monday morning and may vary from week to week. We convert these requirements into units of machine time required for production at Stage One (D_1^i), for each Part Type i, $i = 1, \ldots, n$. Current requirements must be delivered by Friday evening that same week. In addition, projections through week m are provided and converted to the same units ($D_2^i, \ldots, D_m^i$), for each Part Type i. Demand due by the end of the current week is frozen (Sridharan and Berry, 1990) and must be completed with a certain level of confidence. Demand for the next m weeks is known with a high probability, but is also assumed to be frozen.

Other System Randomness

Failures are random events which stop a machine from working. While steps may be taken to reduce the frequency or duration of failures in the system, these issues are beyond the scope of this thesis. In fact, we assume that there are no failures in the system. Aside from system failures, other random events may also affect the system performance. An employee can get sick; a machine operator might get back from lunch 5 minutes late; or the time for producing a part may vary. When of all these events are combined, they can have a significant effect on the system, even if individually unimportant. This variability is also ignored in this thesis.

We stated (Section 1.1) that any production scheduling philosophy must deal with failures and these other sources of randomness. However, we have deliberately assumed away a great deal of the randomness in the system. We do this for model tractability. We hope to be able to extend the methods developed in this thesis for dealing with one simple random phenomenon (setup times) to all sources of variability. In the implementation discussion (Chapter 7) and Conclusion (Chapter 8), suggestions for these extensions are provided. The method presented in this thesis should be capable of dealing with these added dimensions of randomness because of the inherent stability and robustness of feedback control.

Management Influence

Upper level management decisions may have a great impact on the system's capability. For example, decisions related to system layout, shipping schedules, setup frequency targets, etc. are all pre-specified. We assume first that overtime options are available in high-demand periods. Second, we assume the sales department understands the limits of the production system in making its demands. Third, cycle time expectations are in line with system capabilities. Finally, the

maximum allowable inventory levels are sufficient for system efficiency.

These assumptions are critical for the successful implementation of our algorithm. For example, unreasonable expectations of setup frequencies may lead to spiraling losses in system capacity. The same is true if sales departments have unrealistic expectations or if management tries to schedule based on cycle times which the system cannot physically achieve. Management must be willing to keep limited inventory. Inventory is often required to overcome infrequent events such as an up- or downstream machine failure. Casual observations have indicated that many managers have been praised in the short run for reducing inventory, only to cause major production inefficiencies when the bottleneck resource is shut down due to starvation caused by an system failure. While addressing these issues is important, they are beyond the scope of this thesis.

2.4 Scheduling Objectives

The system scheduling objectives were compiled based on discussions with management. These objectives include: low lot sizes, high product mix variability capability, low lead times, low inventories, low backorders, frequent shipping schedules, high flexibility (low setup times) and 100% yield. These objectives were then combined and prioritized. Quality is the first and most important objective. A reasonable balance between flexibility and system utilization is the second objective. Meeting due dates with a low lead time and a high level of confidence is the third objective.

Maintenance of Quality Federal health regulations are very strict, causing quality control to be the paramount objective at the facility. Associated with such quality restrictions is a tremendous amount of documentation and process testing.

The effect of prioritizing quality is that significant, random setup times will remain an important issue which must be incorporated in the scheduling methodology.

Balance Between Flexibility and Utilization *Flexibility* is a measure of a system's ability to produce varying combinations of quantities of part types within the week. We define *utilization* as the time a machine is producing good parts divided by the time the factory is open for production (*i.e.*, utilization is an indicator of the fraction of time the system is doing something useful). Since the machine cannot produce while setting up, the utilization drops as the setup frequency increases. Utilization is not end unto itself. However, usually the higher the achievable average utilization is, the lower the capital equipment requirements are for a given service level.

Given significant setup times (low flexibility) and limited capital equipment prohibits the making of every part type every day. This is because the total time setting up would exceed the total available machine time. In fact, even making each part type each week would severely limit machine utilization. To balance between flexibility and system utilization, we suggest that lots of each part type be made no more than once each week, with shipments of finished parts only on Fridays. This decision is at a higher level than dispatching and will be assumed to be a system constraint in our model development.

Establishing and Meeting Reasonable Due Dates Meeting due dates consistently from week to week is at the third level of priority. For this to be achieved, management must have realistic expectations of the system's capacity in establishing the weekly demand. The maximum amount of time required for production and setups must be less than the number of hours available on the machine, for a given level of confidence. With reasonable expectations by management and careful scheduling, meeting weekly demands is a feasible task. Unrealistic system

expectations can lead to system instability caused by excessive setup changes. This will often lead to drastically reduced system utilization. For example, the more management demands, the less opportunity there is for building ahead. This forces more part types to be made each week, which requires significant setup time and reduces system utilization. This allows even less production of future requirements which further lowers utilization. An endless cycle of utilization loss is established. Our research focuses on this issue of meeting due dates within the constraints of the system.

2.5 Current Approaches to the Problem

Our goal is to schedule production and machine setups in an environment of random demand and significant random setup times to meet set due dates. We assume that quality levels, setup frequency targets and utilization ratios are given. In this section we examine the present methods being implemented to meet production due dates. Observed efforts include increasing demand, simulation, "Fudge Factors" and human decision making. Inherent in all these methods is the assumption that lot sizing is a decision that is dominant over lot sequencing (Section 3.1). In Section 3.4, this premise is challenged.

Increasing Demand

Marine Sergeant-type production managers may be gone from the factory, but vestiges of their tactics still remain. A common strategy for dealing with an under-performing factory is to demand more of the system, hoping that eventually the desired objective will be achieved. Casual observations indicate that this is very poor strategy, resulting in reduced machine utilization and poor morale.

Simulation

With the proliferation of inexpensive computing power, management is now able to gather tremendous amounts of data from the factory floor. The factory has an excellent Computer Integrated Manufacturing (CIM) system in place which allows machine operators and management to instantly access information about the system. Management has developed elaborate simulations, using the available data, to assist in making future production planning decisions.

Simulation, however, has many fundamental flaws which make it very difficult to use in production planning. It often does not accommodate event-driven control policies. Most results of simulations provide long run averages, providing little help in scheduling the system in real time. Good simulations require a long time to program, and the people with the skills to write the simulations are far removed from the process. Finally, simulation is often used to predict future system performance when parameters such as mean time to failure, mean time to repair, etc., are unknown. Poor estimates of these parameters can lead to false conclusions from the simulation results.

Fudge Factors

Current scheduling software algorithms are often based on the calculation of expected or average values for the times of random events (*i.e.*, expected production rates, failure rates, setup time, etc.). Using standard spreadsheet or Gannt chart planning packages, the scheduler makes an estimate of system capacity and designs a production schedule for the facility. Unfortunately, the system is subject to many random events. The actual time for events will often exceed the expected time. This has the effect of forcing the production planner to spend the end of the week scrambling to reconfigure the schedule due to unforeseen events. To avoid this end-of-week turmoil, planners use "fudge factors" which over-compensate for

the amount of time for a particular process step to account for variability. This unneccesarily allocates extra capacity which may often get wasted. This type of strategy coincides with the JIT philosophy (Golhar and Stamm, 1991). However, rather than letting the machine sit idle when the extra allocated time is not required, a method for using the extra capacity for secondary priority items is suggested.

Human Decision Making

The scheduling of a machine is often left to the discretion of the machine operator. He/she has the ability to react in real time to events that occur on the shop floor. It is the intention of this thesis to outline a theoretically justified decision support methodology to aid in this process. Given the above list of priorities and assumptions, the objective of the controller is to schedule current production with very high reliability. The algorithms developed in this thesis schedule the system in real time, allowing production planners to respond to random events.

Chapter 3

Solution Approach

Chapter 2 outlines the characteristics of a dynamic manufacturing environment. System randomness in this environment is assumed to be limited to setup times, which are variable, significant and sequence-dependent. Each Monday morning the facility receives current requirements due the following Friday evening which must be met with a very high level of confidence. In addition, projected future demand requirements are provided with equally strict due dates. Presently employed approaches to scheduling the system include increasing demand, simulations and "fudge-factors." Little account is made for system randomness as it occurs, and casual observations suggest that on-the-fly decisions outperform set schedules. This has lead to an inherent distrust of the effectiveness of set schedules by both upper management and machine operators.

In this chapter, a different methodology, focusing strictly on the system bottleneck, is proposed for scheduling the system. The approach is justified within the context of hierarchical decomposition theory and concentrates on controlling the bottleneck using a two-level method of lot sequencing (Step 1) via a greedy

algorithm and lot sizing (Step 2) via dynamic programming to obtain a closed loop feedback control policy. A description of the hierarchical decomposition approach and key definitions are provided (Section 3.1). The basic premises of the approach are described (Section 3.2). The facility is examined within the context of the decomposition methodology (Section 3.3). Finally, an approach for scheduling production at the bottleneck is suggested (Section 3.4).

3.1 Methodology Descriptions

The concepts of hierarchical decomposition, closed loop feedback, real-time control and part dispatching are essential to understanding our approach to scheduling.

Hierarchical Decomposition *Hierarchical decomposition* breaks down long-term, overall system goals into more refined, short-term, subsystem goals which are more easily modeled, analyzed and controlled. (Burch, Oliff and Sumichrast, 1987; Gershwin, 1987, 1989; Gershwin, Caramanis, and Murray, 1988). The solution to the overall scheduling problem is implemented using a decentralized structure of independently controlled, hierarchically organized modules. Other recent work describing the use of hierarchies include Jones and McLean (1986) and Liberatore and Miller (1985).

Activities in a manufacturing system include producing parts, changing machine configurations and failures. *Events* are points in time marking the beginning or ending of an activity. *Spatial decomposition* is the decoupling of a production process with buffers large enough to prevent events occurring at different portions of the process from affecting one another very often. *Temporal decomposition* is the separation of the control of the system based on event frequencies. The separations

must be such that Level n controllers can treat higher level events as static system conditions and events at lower levels in an aggregate manner (*i.e.*, production may be represented by a flow rate rather than as a set of discrete events).

Violette and Gershwin (1991) provide a framework for a combined spatial and temporal decomposition, the basic structure of which is displayed in Figure 3-1. Higher levels are represented by lower numbers. The actual process is usually represented in the lowest level in the hierarchy (Level 3) by machines (d, f, h and j) and buffers (e, g and i). Larger circles represent buffers with greater capacity. Lower levels in the hierarchy symbolize controllers that deal with higher frequency, shorter duration events. The number of control modules on each level is determined by the buffer sizes in the process. Adjacent machines (f and h) may be controlled by different modules at Level 2 (b and c) if the buffer (g) between them is large enough to allow the propagation of effects of Level 2 events from one machine to the other only very rarely. If the decomposition criteria are met, different modules rarely communicate with one another except through the aggregate information flows passed up and down through the hierarchy.

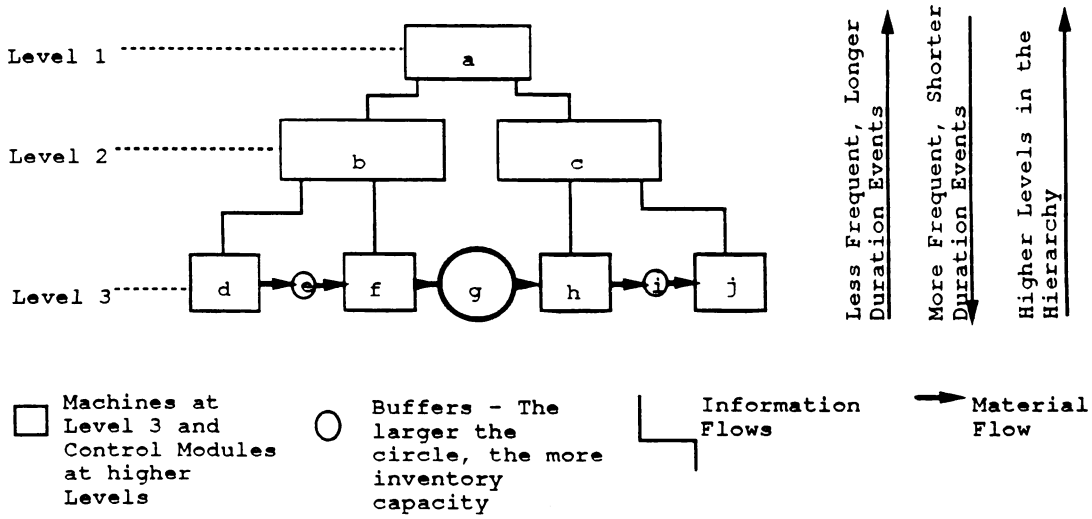

Figure 3-1: Symbolic Hierarchy

Real-Time Closed Loop Feedback Control The *system state* is the critical information about the system required to make a control decision. *Closed loop feedback control* specifies an action to be taken as a function of the system state. Figure 3-2 illustrates the feedback cycle: the system state is determined (*e.g.*, how much time is left in the week, remaining demand requirements, etc.); a control action is implemented based on a measurement of the system and a feedback law; the system state changes in reaction to the control and the environment; and the cycle is repeated. *Real-Time* closed loop feedback control requires that the time to measure the system state and determine an appropriate control action based on this measurement be reasonably small. The time is reasonably small if the action can be determined and implemented before the state of the system changes to any measurable degree.

Part Dispatching *Part dispatching* is the determination of how much and at what time to load material at a particular machine. *Lot sequencing* is the determination of the order of setup changes. *Lot sizing* is the determination of the amount of each part type to produce. Dispatching involves simultaneously dealing with lot sequencing and lot sizing. It is usually dealt with by a control module at a low level in the hierarchy. We use a real-time, closed loop feedback policy for

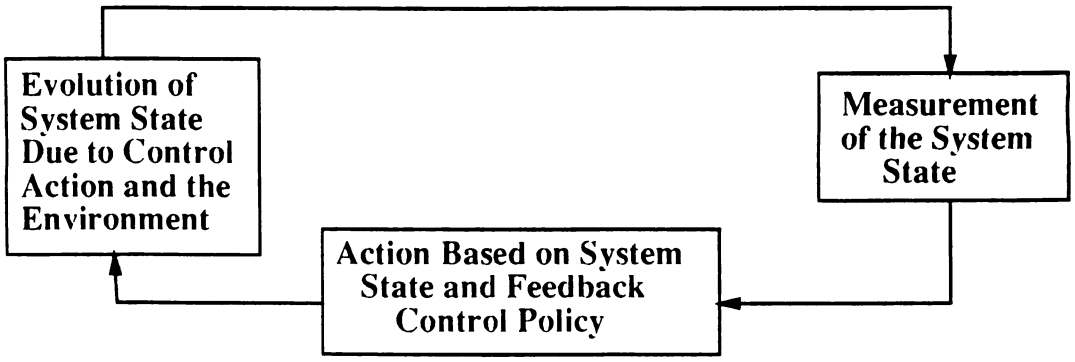

Figure 3-2: A Feedback Loop

controlling part dispatching at Stage One.

3.2 Model Premises

Our method rests on four basic premises. First, given the system conditions, control can be concentrated on a single bottleneck. Second, management only demands production which is feasible in both the short and long term. Third, lot sequencing and lot sizing problems can be formulated separately. Finally, implementing frequently calculated near-optimal control policies is more effective than implementing infrequently calculated optimal policies.

3.2.1 Isolated Bottleneck

A cost analysis of the manufacturing system indicates that Stage One has the highest marginal cost of production. Therefore, the facility would like to maintain the minimum possible amount of this resource, which in turn requires that the resource achieve its maximum throughput. To obtain this objective, we suggest forcing the resource to remain a static bottleneck by providing redundant capacity up- and downstream. We contend that the added expense for this capacity is worth the reduction in system instability caused by random moving bottlenecks and the sophisticated control structures required to manage them as well as the increased throughput at the bottleneck. We justify controlling the overall system by effectively managing the bottleneck by hierarchical decomposition (Section 3.3).

3.2.2 Reasonable Demands by Management

Management frequently assumes that long-term, average production rates may be used for short-term capacity planning. This often leads to poor schedules because of system randomness. During weeks with higher than average machine downtime, the factory will not be able to meet average production levels. The closer demands are to capacity, the higher the probability that demands cannot be met within a given period of time due to unpredictable random events. In an attempt to best satisfy requirements, production managers are often sent scrambling at the end of the week to please the most important customer. This requires frequent setup changes and results in significant losses in utilization. Then, further setbacks are incurred in future weeks in order to fulfill demand for the current week's unsatisfied customers.

Our method forces management to limit their expectations of the system to what it can produce with a high level of confidence under uncertain conditions. This means management must demand less of the system than can be produced on average. Casual observations have shown a great reluctance by management to accept the idea of demanding less of a system. However, in response to their objections, we showed them how the extra allocated time is not wasted by being used to build larger lots (Chapter 5). This maximizes machine utilization in the current week and frees up capacity in future weeks.

3.2.3 Separation of Lot Sequencing and Lot Sizing

We assume that lot-sequencing and real-time, lot-sizing algorithms can be formulated separately in a hierarchical fashion (Section 3.4.1). Logistic constraints often prevent real-time sequence optimization. For example, significant setup times prevent the scheduling of very small lot sizes due to limits on the maximum feasi-

ble setup frequency of once per week. Raw materials and finished goods are often shipped according to fixed schedules to avoid excessive shipping costs, also limiting scheduling. Assuming that each part type is only made once per week and demand is within capacity for a certain level of confidence, we maintain that the setup sequence should be designed to minimize total setup time (Section 3.4.2). Then, using this sequence, lot sizes can be determined in real time (Section 3.4.3).

3.2.4 Efficiency of Frequent Near-Optimal Control

Sophisticated models can often be formulated for the most complex manufacturing systems. However, to generate an optimal control policy from these models can take a prohibitively long time. In fact, in stochastic environments, the problem may even be unsolvable. Near-optimal solutions that can be calculated and implemented very frequently provide a better alternative to infrequently implemented optimal policies. Even if the controller occasionally chooses a non-optimal action, it can recalculate so frequently that the system directs itself back on track almost immediately. This is due to the inherent stability of feedback control.

3.3 System Hierarchy

Figure 3-3 is a diagram of the management structure at the facility. We assume that this structure is given and will not change upon the implementation of the algorithms developed in this thesis. The management hierarchy is structurally similar to a control hierarchy. However, an arbitrarily constructed decomposition will not necessarily be efficient for control purposes. Therefore, in this section, we show that an isolated control strategy for Stage One as indicated by the management structure is consistent with the assumptions of decomposition, and consequently

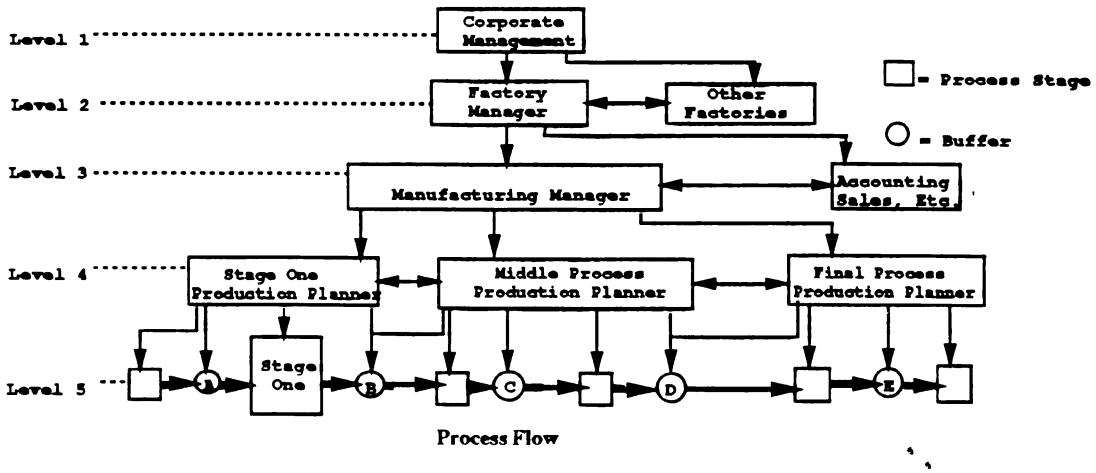

Figure 3-3: Management Structure

that local control of Stage One can be justified.

Spatial Decomposition

Part dispatching is a frequent event in our system. Therefore, part dispatching is controlled at a low level in the hierarchy. To be consistent with the spatial decomposition assumptions (Section 3.1), independent control of Stage One is feasible only if the buffers separating Stage One from the rest of the system are sufficiently large to prevent low level events occurring upstream and downstream from affecting dispatching at Stage One.

The upstream process consists of raw material pre-processing. The buffer prior to Stage One is assumed to be unlimited and always has sufficient raw material so that Stage One is never starved (Section 2.2). Given the excess downstream capacity, only a very small buffer is needed before Stage Two to prevent the effects

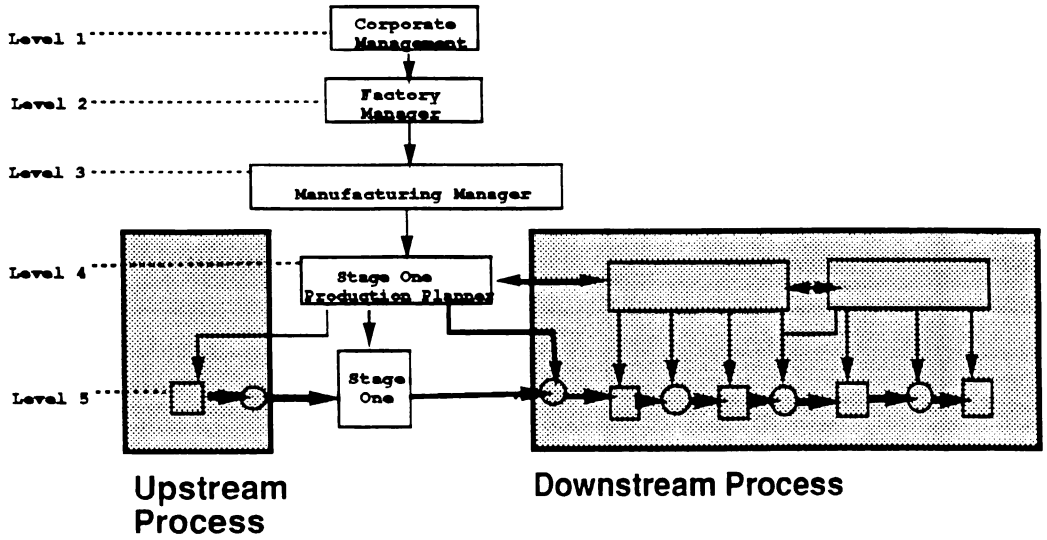

Figure 3-4: Spatial Decomposition

of downstream events from affecting Stage One. Upstream activity may be treated as a virtual machine with unlimited resources. Downstream may be treated as a virtual machine with unlimited capacity. These assumptions are consistent with the assumptions of spatial decomposition. Therefore, for this facility, it is reasonable to isolate the control of Stage One (Figure 3-4) because part dispatching is rarely affected by the events in other parts of the system.

Routing Decomposition

Stage One (Figure 3-4) actually represents a set of similar machines. Since the system has a deterministic routing policy (Section 2.2) and the previous argument for spatial decomposition holds for each of the individual Stage One machines, we may focus our algorithm development on one isolated machine. This algorithm may then be employed independently and simultaneously at all the Stage One machines.

Temporal Decomposition

Table 3.1 lists system events and their frequencies. To fulfill the assumptions of temporal decomposition (Section 3.1), the system conditions and the targets established by the Manufacturing Manager (upper level) for the Production Manager (lower level) must relate only to events which occur significantly less frequently than the loading of parts and changing of setups (events controlled at the dispatching level). In addition, sufficient information for capacity planning must be transmitted upward so management can set achievable production targets. If these assumptions are fulfilled, higher level control actions and events may be assumed as static system conditions dictated by an upper level black box (Figure 3-5) and part loading may be treated as a flow by the dispatching controller.

Table 3.1 indicates that dispatching level events (*e.g.*, part loading and setup changing) occur at least once per day, on average. In the development of our model (Section 3.4), we may assume that demand is frozen, deliveries occur once at the end of the week, staff and capital are fixed, machine utilization target are set, all scheduled downtime has been accounted for and probability distributions for random variables (*e.g.*, setup times) are fixed. Table 3.1 indicates that changes to

event type	*average frequency**
part loading	1 per minute
setup changing and lotsizing decision	3 per day
finished goods shipment	1 per week
demand change	1 per week
scheduled downtime decision	1 per week
system parameter changes	1 per week
setup frequency decision	1 per 6 months
delivery and demand arrival schedule decision	1 per 6 months
staffing or capital purchase decision	1 per 6 months

*Orders of magnitude.

Table 3.1: System Events and Their Frequencies

each of these system constraints occurs at a frequency substantially less than the frequency of setup changes and part loading and may be treated as static system conditions in our model. Lower level controllers transmit achieved production quantities and adjusted random variable parameter estimates sufficiently often so that the manufacturing manager control module may set reasonable weekly production targets. According to the hierarchical decomposition methodology, these conditions imply that part dispatching control may be decomposed from upper level controllers.

Table 3.1 also indicates that part loading is significantly more frequent than setup changes. Therefore, the discrete loading of parts is approximated and modeled as a flow. Consequently, dispatching control is isolated in the Production Planner and Stage One modules in Figure 3-5 and we proceed to develop the system scheduler (Section 3.4) by focusing on the bottleneck at Stage One.

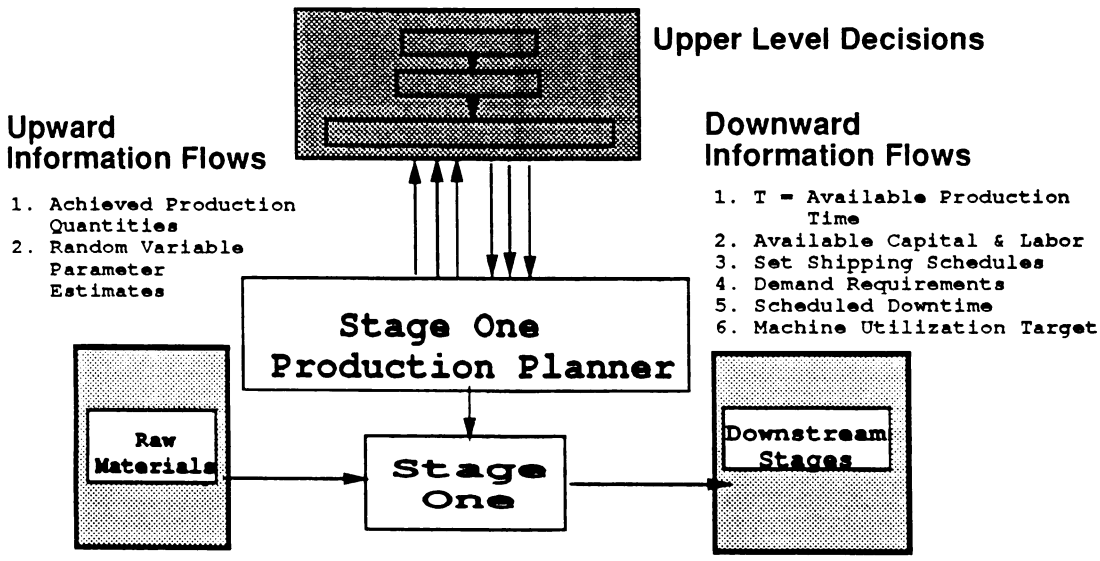

Figure 3-5: Temporal Decomposition

3.4 Part Dispatching Methodology

The system must produce variable quantities of part types demanded each week, on one machine, with a very high level of confidence. Once requested, demand is fixed and does not change. Setup times are significant, sequence dependent and random with known distributions. The facility may not set up for a particular part more than once per week. However, even producing every part type once in a week would require more setup time than is available on average. Future demand for a limited number of weeks is known. Labor, capital, available production time, setup time distributions and shipping schedules are all fixed within the time frame of our model. In addition, the cost of inventory is negligible and machine capacity is expensive.

We assume that there are no failures and that production rates are deterministic. We also assume that management has a good capacity model and that weekly production targets, if scheduled properly, are achievable with a high level of confidence. Finally, we assume that any requirement for a part type for any week may not be subdivided unless it is the last part type produced in the week (*i.e.*, one can not make a fraction of an order one week and then a fraction the next). This assumption is not critical, but it makes the problem formulation easier to understand.

Our method involves establishing a two-level hierarchical dispatching policy (Section 3.4.1). A lot-sequencing algorithm is established (Section 3.4.2) which is then used as input to a lot-sizing algorithm (Section 3.4.3).

3.4.1 Dispatching Hierarchy – Two-Level Method

A hierarchy, but more refined is established for controlling part production dispatching at the bottleneck. The hierarchy separates part dispatching into two functions at different levels. The policy is such that lot sequencing (Level 1) dominates lot sizing (Level 2) (Figure 3-6). In fact, at least one more level exists below lot sizing which controls the loading of individual parts. However, at this level, production rates (Section 3.3) may be met by using a staircase policy (Gershwin, 1989), and for our purposes, no interesting phenomena occur at lower levels.

The concept of a hierarchy for low level production planning is not new (Bitran and Hax, 1977). However, the traditional precedent has been to let lot sizing dominate lot sequencing. For example, casual observations of many planning operations indicate that corporate management establish lot sizes (traditional *economic lot sizing problem*, Section 5.2) and due dates at a high level, leaving the task of lot sequencing (traditional *earliness/tardiness* problems, Section 4.2) as the responsibility of the factory operators. Without the current communications technology, it

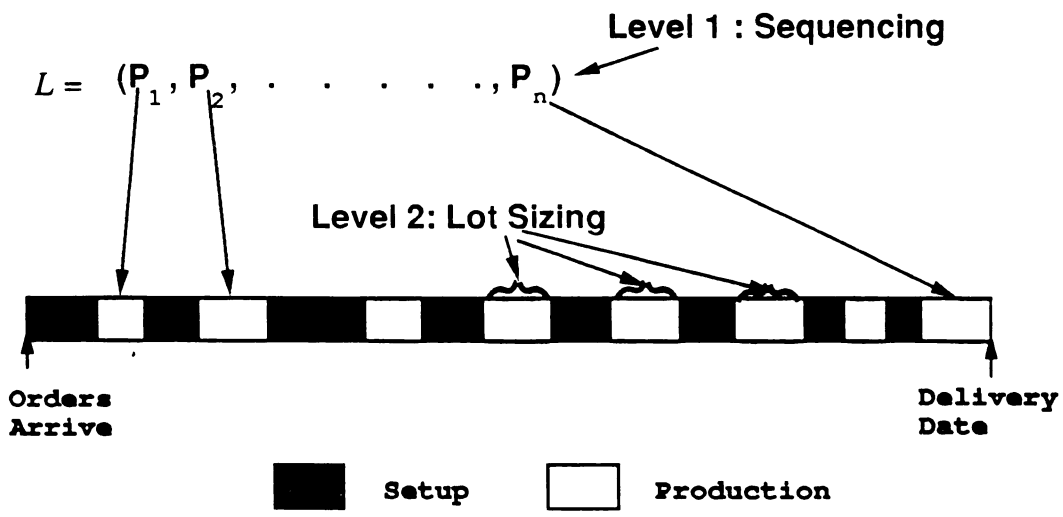

Figure 3-6: Two-Level Method

was too hard to provide real-time demand data to the factory required for real-time lot sizing. In addition, the computing power did not exist for the computationally intensive calculations required to process the information. Therefore, most previous work in this area has always assumed lot sizing as the dominant decision over lot sequencing. Today, both the computing and communications technology exist to allow factories to make setup and production decisions in real time.

While our scheduling method assumes that the lot-sequencing and lot-sizing problems may be solved independently (Section 3.2.3), in reality, secondary effects related to the most efficient building of inventory violate this independence assumption. For example, it may be more advantageous to sequence production such that the part types for which inventory is more desirable are produced later in the week when the variability of future activities in the week is at a minimum. It may also be more efficient to end the week with a specific machine configuration. However, we overlook these secondary effects.

3.4.2 Setup Sequencing – The $\mathcal{L}$ *List*

Each week, a list of part types and their associated required quantities due by common date are provided. These different part types establish the minimum number of required machine configurations for that week. The common due date implies that as long as the parts are produced by the due date, the machine configuration sequence will have no bearing on the cycle time for the required production. This allows us to concentrate on the sole objective of minimizing total expected setup time when sequencing.

The controller is provided with the setup time distributions and the initial machine configuration. The algorithm then sequences the machine configurations into an $\mathcal{L}$ *List* of part types (P^i, $i = 1, ..., n$) which minimizes expected total setup

time (Chapter 4). This sequencing problem is often formulated as a stochastic Traveling Salesman problem (Johnson and Montgomery, 1974), which is generally very difficult to solve (Section 4.4). An embedded hierarchical structure (Section 4.3) is explored within the setup time distributions. Using this structure, reasonably good $\mathcal{L}$ Lists may be obtained quickly by a *greedy algorithm* (Papadimitriou and Steiglitz, 1982, pgs. 279). While the greedy algorithm does not guarantee an optimal solution, the suboptimal solution it provides can be obtained in a matter of seconds. In addition, it is possible to bound the difference between this suboptimal solution and the optimal one (Section 4.5). Once the machine configuration sequence is established, it may be used as an input to a lot-sizing algorithm which reacts to system events (*e.g.*, machine failures, yield problem, etc.) in real time.

3.4.3 Dynamic Lot Sizing

Given the $\mathcal{L}$ List, the second step in the scheduling method is to establish lot sizes in real time. Figure 3-6 shows a Gannt chart summary of a typical week of production, where the horizontal distance represents the time spent on a particular activity (*i.e.*, black bars indicate time setting up and white bars, production run time). Lot sizes are determined for each part type in the $\mathcal{L}$ List upon the completion of each random event in real time. Adjustments are made to the schedule by altering the lengths of the white bars in Figure 3-6 while maintaining the part type sequence of the $\mathcal{L}$ List.

As events occur (*i.e.*, setups are completed, jobs are completed, etc.), the actual time required to complete an activity will differ from the time originally allocated to the activity. For example, upon the completion of a setup, the time required for the setup, which was an unknown quantity, becomes a known quantity. To account for variability, more time is normally allocated to setups than is required on average. The difference between the actual time for a setup and the time

allocated becomes available upon the setup completion. With the occurrence of each event, the amount of accumulated time for discretionary production changes and is equal to the previous amount of accumulated discretionary time plus or minus the difference due to the completion of the most recent activity (*e.g.* a setup). This new information may then be used to establish a more efficient schedule for the future.

Chapter 5 focuses on the real-time decision how to schedule production by deciding in real time how large to make each lot. The problem is modeled as a dynamic program. From the characterization of the optimal solution, a sub-optimal heuristic is proposed which can be calculated very quickly and frequently.

Chapter 4

Setup Hierarchy

As discussed in Section 3.4, a two-level process (lot sequencing and lot sizing) is suggested for scheduling the bottleneck of the manufacturing facility described in Chapter 2. In this chapter, the lot-sequencing algorithm, which is based on a setup hierarchy structure, is developed in detail. The algorithm maximizes the amount

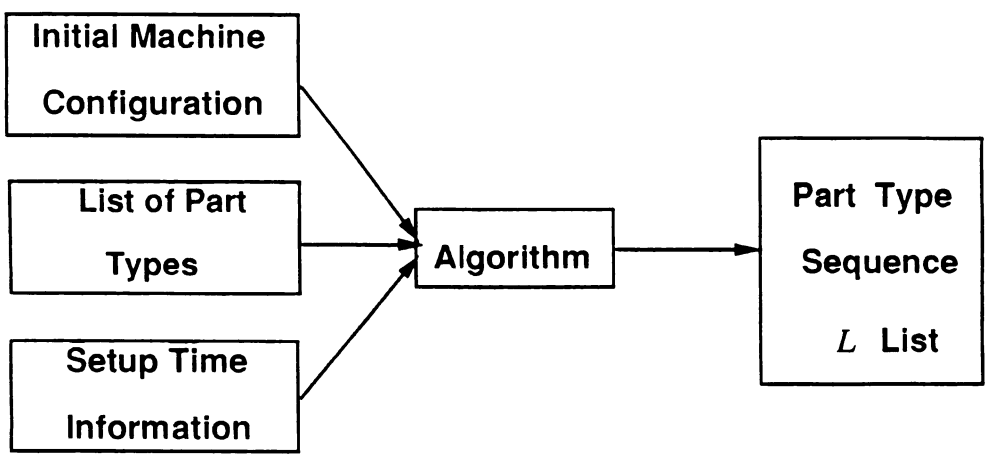

Figure 4-1: Part Type Sequencing Algorithm Flow

of available production time by minimizing the total setup time. Total setup time is a variable cost which may occupy more than half of all available machine time. All other costs associated with setting up are fixed within a given week. Total setup time is therefore the only cost required in a model designed to minimize the total cost. The model input data is the initial machine configuration, a list of all part types produced by the machine and setup code information representing the machine configuration required to produce each part type (Figure 4-1). These setup codes require significantly less computer storage than the matrix of sequence dependent setup times normally required for setup sequencing. The output is a sequence of part types (the $\mathcal{L}$ List, which is used as input to the real-time dynamic lot-sizing algorithm described in Chapter 5).

Section 4.5.1 shows that when the setup times have a hierarchical structure (Section 4.3), a greedy algorithm provides a sequence which results in the lowest total setup time in a given week. In a hierarchical structure as defined here, the setup time distributions for all Level i (Section 4.3.1) changes are identical. However, in the case when there is limited variability among the distributions, the greedy algorithm may still provide a solution within a reasonable bound of the lowest possible total setup time (Section 4.5.2). For the prototype in Chapter 7, this hierarchical decomposition is unnecessary to generate the $\mathcal{L}$ List because a simpler *ladder structure* (Chapter 7) can be exploited for sequencing the experimental machine, which only produces seven part types. However, for a full-scale implementation, the more sophisticated algorithms often require data that cannot be obtained and/or maintained. Such cases benefit from the hierarchy.

A literature review of the sequencing problem is provided (Section 4.1), followed by a list of definitions and notation for establishing the algorithm (Section 4.2). The hierarchical setup structure is described (Section 4.3). A mathematical abstraction of the sequence-dependent setup problem is developed (Section 4.4). A greedy algorithm for solving this problem is suggested and proven to provide a sequence which results in a total setup time within a determinable bound of the

lowest possible time (Section 4.5). A discussion of the practical benefits of the policy are presented (Section 4.6). Finally, the chapter concludes with summary remarks (Section 4.7).

4.1 Sequencing Literature Review

Sequencing is the ordering of parts to be loaded into a system. Melnyk and Carter (1986) discuss the role of sequencing and its relation to the general dispatching problem. The primary objective of sequencing is to load jobs with different priorities and service times into the system such that some performance measure is optimized. The earliness or lateness of job completions, the *makespan* (the time from beginning production of the first part to the completion of last part) and/or *flowtime* (the sum of the processing times of all the parts) are among many possible measurements. The one-machine sequencing problem looks at n jobs on one machine. Baker and Scudder (1990) review this problem with associated earliness/tardiness penalties. In our factory, there is a very heavy tardiness penalty.

In this thesis, we look at scheduling *bottleneck machines* which have less capacity (Section 2.1) than other machines in the manufacturing system. Scheduling bottlenecks is a special case of single machine scheduling. Bottleneck management is the primary thrust of OPT software (Goldratt and Cox, 1986). Recent work on lot-sizing in the presence of bottlenecks is presented in Chang, Matsuo and Sullivan (1989). Others look at the handling of temporary or moving bottlenecks (Pence, Megeath, and Morrell, 1990). The capacitated hierarchical flow control methods of Gershwin (1989) also address this issue.

Most machines, including those at the bottleneck of our system, are not capable of immediately starting production of a new part type without some type of *setup* (Section 2.1). While a papers exist on scheduling in the presence of setups

(Monma, 1989; Tang, 1990; Dilts and Ramsing, 1989; Trigeiro, 1987, 1989; Galvin, 1987; Charles-Owaba and Lambert, 1988; Trigeiro, Thomas and McClain, 1989), very little is available on changing setups in response to system events. Recent work on real-time setup policies is presented in Srivatsan and Gershwin (1990), Connolly (1992) and Sharifnia, Caramanis, and Gershwin (1991). However, these papers assume deterministic setup times and constant, continuous demand rates. By contrast, we assume due dates are discrete and setup times are random. In addition, even if this research is extended to include randomness, the methods become much less tractable for many part types. Our method is less sensitive to the number of part types being produced. All these real-time methods are similar in that they use a hierarchical decomposition approach, which our paper also uses.

We look at setup sequencing at a stationary bottleneck with random sequence-dependent setup times (Section 2.1). The sequencing issue may be treated as either a knapsack-type problem (Papadimitriou and Steiglitz, 1982) when capacity is an important constraint or a traveling salesman-type problem (Johnson and Montgomery, 1974) when capacity is not an issue. While the literature on these subjects is vast, almost all of it assumes that lot sizes or job sizes are given when the sequencing problem is solved by the factory (Section 3.4.1). We assume that sequencing should dominate lot sizing. That is, we first determine the sequence in which to produce part types and then determine how much of each type to make while maintaining this sequence. The advantage of this approach is that management has the ability to adjust capacity by allocating overtime or contracting out excess work out while lot sizes can be determined in response to system events. Since capacity is eliminated as a constraint in our sequencing problem, we can look at the problem as a traveling salesman-type problem (Section 4.4). However, when significant, random setup times are incorporated, this problem becomes impossible to solve in a reasonable amount of time.

We approach the problem by identifying a special hierarchical structure within the set of setup time distributions (Section 4.3). Similar work is described in

terms of major and minor setup and part type families in Tang (1990) and Taylor and Bolander (1986). However, our work extends these concepts to systems with random, sequence dependent, setup times and multiple levels of part type families. In addition, our algorithm is coordinated with a real-time lot-sizing algorithm (Chapter 5) and incorporated in an overall control hierarchy.

4.2 Notation and Definitions

In this section, part type notation is explained. In addition, definitions relating to special characteristics of setup time distributions are provided.

Part Type Notation Non-specific part types are identified by capital letters (*e.g.*, Part Type A, Part Type B, etc.). Each specific part type is identified by $P_{a_1,\cdots,a_N}$. The subscript identifies the machine configuration required to produce the part type. Characteristic or attribute i has k_i different possibilities. This classification scheme is the foundation for the hierarchical setup structure.

Table 4.1 illustrates this classification scheme. It depicts a set of four machine configuration characteristics similar to those on the system bottleneck. These four characteristics are: the configuration required for processing each different raw material (RM), the die used for punching (DIE), a depth setting (DEP) and a length setting (LEN). For example, one of four raw materials arrives at a machine. The machine loader must be set so that it may receive the raw material. The part is then punch pressed by one of two dies. The machine operator is required to manually switch dies, which may take a significant amount of time. The part is punched to one of three depths. Changing depths requires a machine adjustment by the operator. Finally, the part is cut to one of two lengths. Changing lengths requires a quick machine calibration. Each part type is identified by the four-dimensional

a_1	RM	a_2	DIE	a_3	DEP	a_4	LEN
1	23	1	6345	1	2	1	4
2	25	2	6385	2	5	2	6
3	27			3	7		
4	29						

Table 4.1: Machine Configurations

vector indicating the machine configuration required for its production. For example, $P_{2,1,3,1}$ requires raw material 25 to be fed into die 6345, at a depth of 7 units and a length of 4 units (Table 4.1).

Setup Times Let τ_{AB} be the time to change the machine from making Part Type A to making Part Type B. It is the time from making the last good A part until making the first good B part. The following definitions describe special setup distribution environments possible for a set of part types Q:

Total Setup Sequence Independence Q has *Total Setup Sequence Independence* (TSSI) if the probability distributions of setup times do not depend on the part types involved in the machine configuration switch.

Partial Setup Sequence Independence Q has *Partial Setup Sequence Independence* (PSSI) with respect to a subset R of Q if it has TSSI among the part types in R.

In practice, setup time distributions among machine configurations are invariably different. For example, in injection molding, it takes less time to switch to a material requiring a higher temperature than a lower temperature because it takes longer for the molds to cool down. However, although variability is inevitable, these concepts still provide useful descriptive terminology for future reference.

4.3 The Hierarchical Setup Structure

In this section, a setup structure based on a hierarchy in the setup time distributions is developed. Rather than beginning with a formal definition of the hierarchical structure (Section 4.3.2), a description of the setup structure of the bottleneck of the system is described (Section 4.3.1) as an example of a setup hierarchy.

4.3.1 Setup Structure

As shown in Table 4.1, there are four possible attributes which change during a setup at the bottleneck. Each attribute is associated with a different level in the setup hierarchy. Table 4.2 indicates the time required to change attributes and is based on informal observations over a six week period. These times and other parameters in this thesis are in standard units left unspecified for reasons of confidentiality. The magnitude of the expected setup times at least doubled from level to level (*i.e.*, $2(MTTS_{i+1}) \leq MTTS_i$). In addition, the standard deviation (σ_i) of the time to change an i level attribute was on the same order or less as the expected time required for an $i+1$ attribute switch (*i.e.*, $\sigma_i \leq MTTS_{i+1}$).

Much of the work associated with changing one attribute (*e.g.*, paper work, adjustments, special technicians) need not be repeated for lower level (higher level number) attribute changes during a given setup. Therefore, the time required for the setup is dominated by the configuration change requiring the greatest amount of time – what we call the *dominant attribute*. In addition, dominant attribute change times usually incorporate changing all lower level attributes. Therefore, the time of any setup is approximated by the time required for changing the dominant attribute. The setup structure must have higher level (lower level number) attributes always dominate lower level attributes. For example, suppose the new

Level (i)	Associated Setup	Attribute	$MTTS_i$	σ_i
1	Major Setup	RM	240	60
2	Minor Setup	DIE	90	30
3	Major Adjustment	DEP	45	15
4	Minor Adjustment	LEN	15	5

Table 4.2: Setup Parameters

configuration requires both a raw material change and a die change. Raw material change is the dominant setup, requiring 240 time units. The simultaneous switching of both characteristics only requires 240 time units and not 330 time units. This leads to a natural hierarchical structure of the setups (Figure 4-2).

Our methodology associates setups with the dominant attribute change required for the setup. The facility can thus divide its setups into four categories: 1) Major Setups (Level 1) in which the dominant attribute is the raw material required (RM), 2) Minor Setups (Level 2) in which the dominant attribute is the required die (DIE), 3) Major Adjustments (Level 3) in which the dominant attribute is the required depth (DEP) and 4) Minor Adjustments (Level 4) in which the dominant attribute is the required length (LEN) (Table 4.2).

To switch between two machine configurations, a path between them on Figure 4-2 must be established. Each horizontal segment on the path is associated with the attribute labeled to its left which changes during the setup. The dominant attribute is the highest horizontal segment on the shortest path between two part types (or the lowest level of common ancestry of two part types) in Figure 4-2. As stated above, the dominant attribute determines the setup time. For example, to switch from P_{1111} to P_{1112} requires only a minor adjustment of the length (LEN) which takes 15 time units on average. However, to switch from P_{1111} to P_{1222} requires a Minor Setup (DIE) which takes 90 time units on average. Switching from P_{1111} to P_{4121} requires 240 time units, P_{2221} to P_{2231} requires a 45 time units and P_{4121} to any other part type requires 240 time units (Table 4.2).

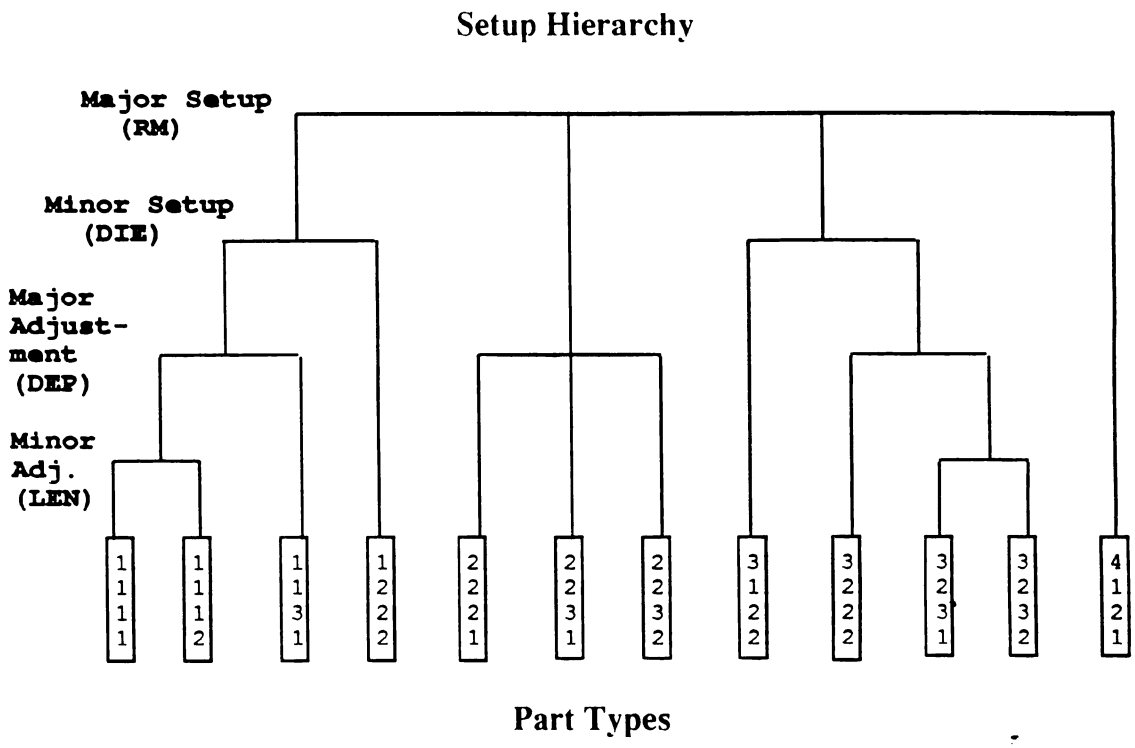

Figure 4-2: Setup Hierarchy

4.3.2 Hierarchy Definition

With the description of Section 4.3.1 in mind, a formal definition of the setup hierarchy can be established. A *hierarchical setup structure* has the following characteristics:

1. Part types are classified by an N-attribute vector denoted in the subscript of the part type (Section 4.2).

2. The expected times to switch various attributes are markedly different from one another. In other words, $\overline{\tau^1} \gg \overline{\tau^2} \gg \cdots \gg \overline{\tau^N}$, where $\overline{\tau^I}$ is average time to complete an I level attribute change.

3. Setups exhibit a *pseudo PSSI* (Section 4.1) defined as follows: $\overline{\tau_{AB}} \approx \overline{\tau^I}$ for all Part Types A and B where I is the level of the dominant attribute associated with a switch from A to B.

4. Standard deviations of i Level setup times are on the same order of magnitude as the expected value of Level $i+1$ setup times or smaller.

We will now use this structure in formulating a mathematical description of the sequence-dependent, setup sequencing problem.

4.4 Traveling Salesman-Type Formulation

A model is developed for the sequence-dependent, setup sequencing problem. First, the system assumptions are detailed (Section 4.4.1). Second, a general formulation of the problem is provided (Section 4.4.2). Finally, the hierarchical structure is used to modify the general problem formulation (Section 4.4.3).

4.4.1 Assumptions

We assume the following in formulating our model:

- All the assumptions already described remain valid. These include: deterministic part routing, excess capacity downstream (*i.e.*, no blocking), present requirements (other than that which is satisfied with existing inventory) within capacity, demand and failures within normal bounds not requiring higher level decisions, as well as all other assumptions imposed by higher levels in hierarchical systems (Section 3.3).

- All τ_{ij} are constants (*i.e.*, there is no variance in setup times) and rely only i and j.
- Processing times are independent of the setup sequence and independent of the part being produced.

In addition, the terms *machine configuration* and *part type* are henceforth used interchangeably due to the one-to-one correspondence between them in the model.

4.4.2 General Formulation – Setup Space

Figure 4-3 portrays a high dimensional space, where each node represents a different machine configuration and the distances between nodes represent setup times. Between each pair of nodes is a pair of directed arcs, but they are not drawn. The *arc distance* (Figure 4-3, b) or arc cost (setup time) associated with

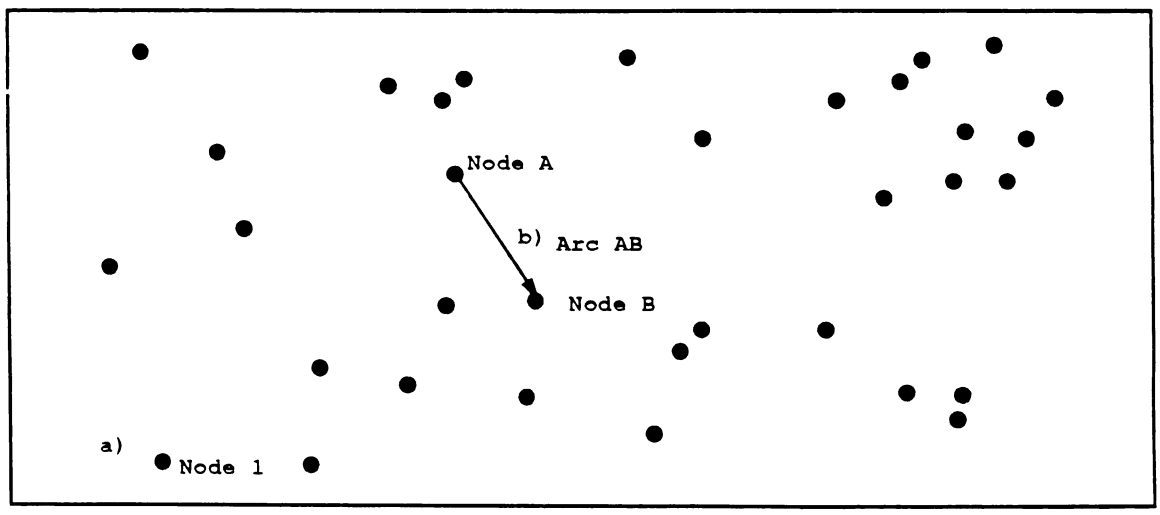

Figure 4-3: Graphical Representation of Minimum Setup Time Problem

each pair of nodes (machine configurations) is represented by τ_{AB} (for nodes A and B) where τ_{AB} is not necessarily the same as τ_{BA}. Given n total part types, the time required for all possible machine configuration changes can be represented by an n by n matrix $\mathcal{T}$. This can be very large, but we show that the hierarchical structure greatly reduces these data requirements. The objective is to find the shortest sequence containing all the nodes in the space. This would be equivalent to finding a setup sequence which has the minimum total setup time. The sequence must begin at a specified initial node (Figure 4-3, a) and the total cost of the sequence is the sum of the all the τ's associated with each segment in the path. This problem is known as the *sequence-dependent setup sequencing problem.*

Although this problem does not require that the sequence finish at a specified node, it is not difficult to show that it is equally complex to solve this problem regardless of whether this constraint is imposed. With this added constraint, the problem is is equivalent in difficulty to the Traveling Salesman Problem, which extremely hard to solve (Johnson and Montgomery, 1974). If instead of constants, the values of the $\mathcal{T}$ matrix are random variables, the problem becomes a stochastic integer programming problem, an even harder problem to solve. In the most general case, systems with even 25 nodes could require an infeasibly long period of time to solve for optimality.

4.4.3 Hierarchical Formulation

When setups have the hierarchical structure discussed in Section 4.3, the setup space takes on characteristics similar to those displayed in Figure 4-4. The machine configuration nodes tend to cluster into families which in turn cluster into larger families and so on. Each dot (Figure 4-4, a) represents the set of machine configuration nodes (Figure 4-4, b) in which only a minor adjustment is required to switch between any two nodes represented by the dot. A major adjustment is

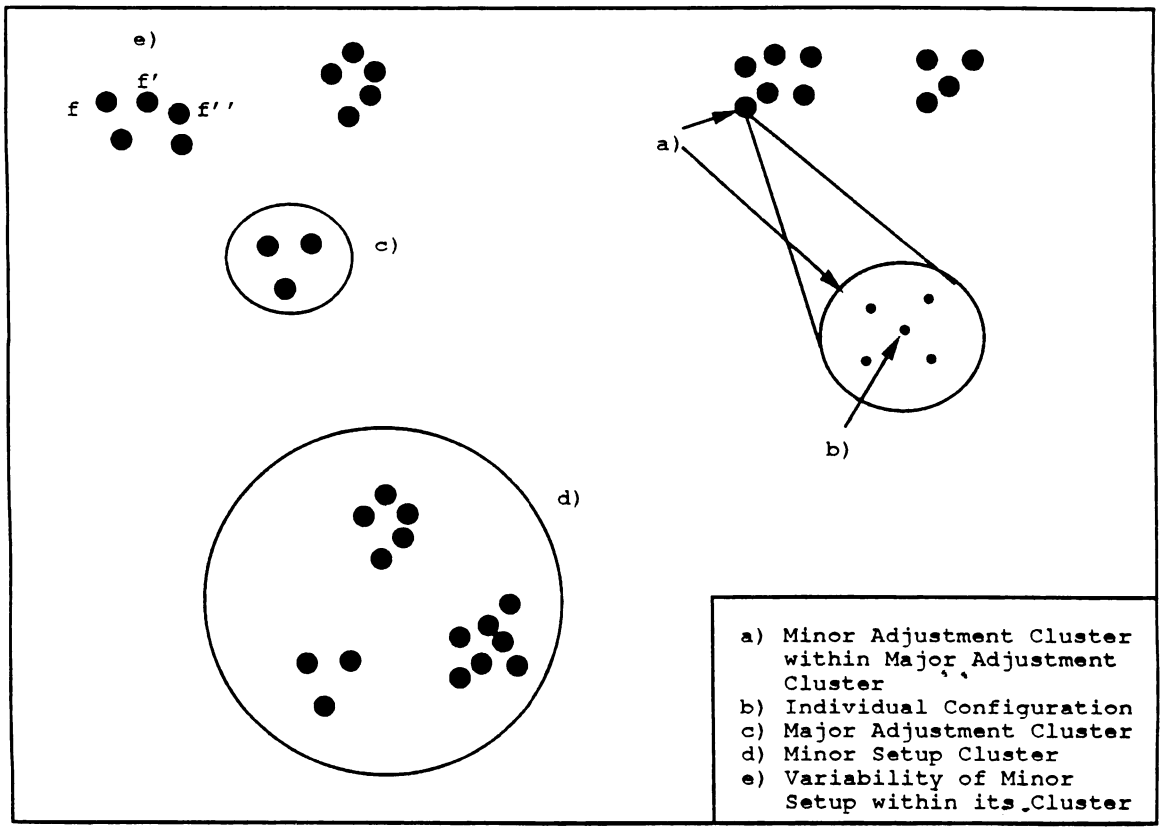

Figure 4-4: Hierarchical Setup Space

required to switch among a dots within a cluster of dots (Figure 4-4, c). A minor setup is required to switch among different clusters of dots. Finally, a major setup is required when switching from a cluster of clusters to another cluster of clusters (Figure 4-4, d).

Since this thesis concentrates on dealing with random setup times, variability is an important issue. In the formation of the hierarchy of clusters, the limits on the standard deviation of the τ's at any level becomes crucial in maintaining the clustering structure. Actual τ's associated with conditions during a specific week may vary (*e.g.*, a node, dot or cluster may move from f to f' to f", Figure 4-4, e). However, the movement is restricted such that the structure of the overall clustering configuration is maintained.

4.5 The Algorithm

We now suggest using a greedy algorithm (Section 4.5.1) to find the solution to the hierarchical setup problem defined in Section 4.4.3. As nodes are added to the sequence, they become no longer eligible to be used later in the sequence. The greedy algorithm is shown to be optimal for a two-level, deterministic setup hierarchy. This type of hierarchy is indicative of the setup structure in many manufacturing environments. In practical applications, a number of setup levels may be required in the hierarchy (such as our facility, which has four levels). Similar proofs to those presented may be used to obtain closely related results for multiple level systems.

A proof is provided for a two-level hierarchy (Section 4.5.1), followed by a discussion of bounds and multiple level extensions (Section 4.5.2). Finally, a formal description of the algorithm is presented (Section 4.5.3).

4.5.1 Optimality Proof

Define a two-level hierarchical system (Figure 4-5) on the digraph $D=(V, A)$ with set of nodes V and set of arcs A. Let h be the cost of a *path* (Papadimitriou and Steiglitz, 1982) traversing each node in D exactly once. Define a *greedy algorithm* as an algorithm which finds a sequence among of set of V nodes by continually choosing as the next node in the sequence, a node j which has the lowest cost arc τ_{ij} between itself and the last node i chosen in the sequence (Figure 4-3). Assume:

Assumption 1 *Arcs in A fall into one of two categories differentiated by the cost associated with the arc. The cost of a Type 1 arc is $k + c$ where $c > 0$ and the cost of a Type 2 arc is k.*

Assumption 2 *Let a Level 2 Cluster, henceforth called a "cluster", be defined as a set of nodes in which arcs between all pairs of nodes in the cluster are of Type 2 and all arcs between a node in the cluster and a node outside the cluster is of Type 1.*

Assumption 3 *Let every node be an element of one and only one cluster, and let there be i clusters, where Cluster j contains C_j nodes, $j = 1, \ldots, i$.*

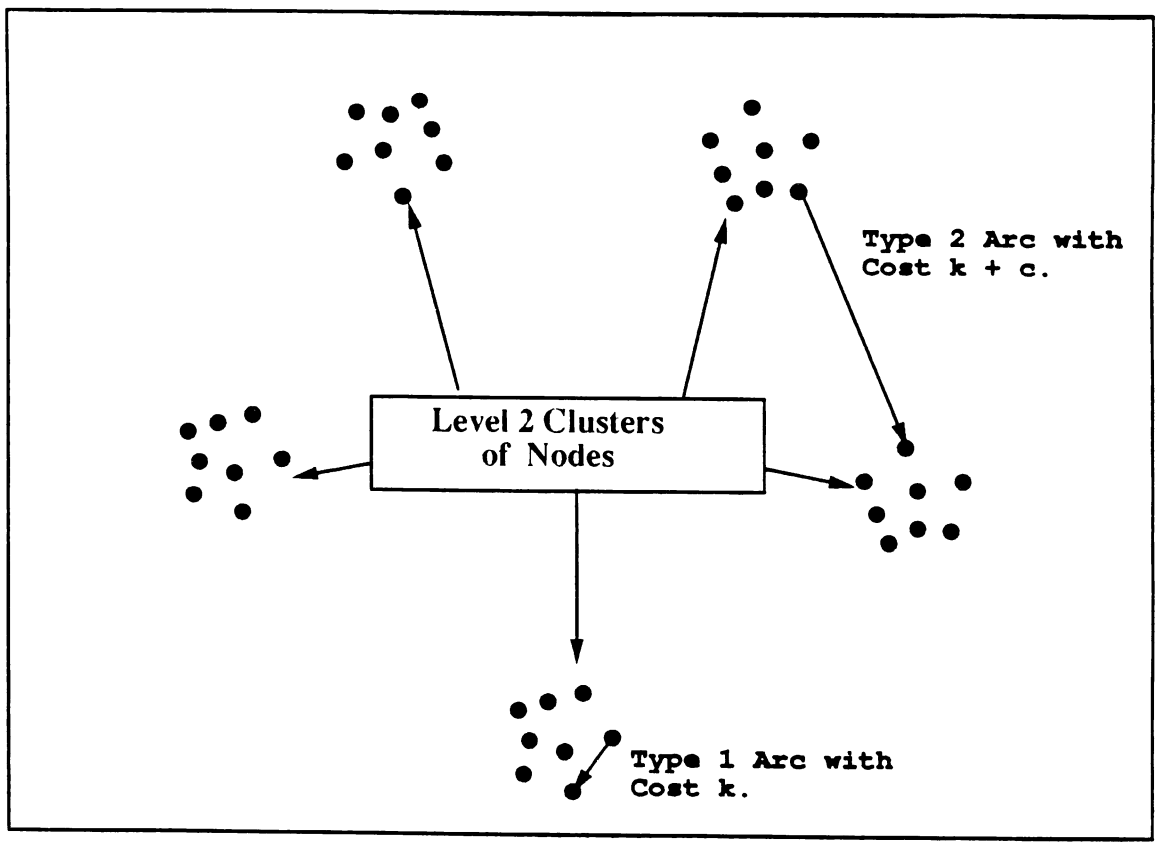

Figure 4-5: Two-Level Hierarchical System

Theorem 1 *The shortest path (smallest h) for a two-level hierarchical system (TLHS) can be found using a greedy algorithm.*

Proof: The proof establishes a lower bound for the cost of the TLHS and shows that the greedy algorithm provides the identical cost, thus proving optimality.

Lower Bound To obtain a lower bound on the cost h, we first determine the minimum number of inter-cluster (Type 1) arcs required in the path. Since each node must be included in the sequence and each cluster must have at least one node, at least one node in every cluster must be included in the sequence. Therefore, there must be at least i-1 Type 1 arcs. This would cost at least (i-1)(k+c). In addition, the lower bound must include the cost of a minimum number of intra-cluster (Type 2) arcs. To determine the number of Type 2 arcs, we note that there must be at least one arc in the sequence emanating from each node, except for the last node in the sequence, and i-1 of these arcs are Type 1 arcs already accounted for. Therefore, there is a minimum of $(\sum_{j=1}^{i} C_j) - 1 - (i-1)$ or $\sum_{j=1}^{i} C_j - i$ Type 2 arcs in the path. Therefore, a lower bound on h (h_{LB}) is:

$$h \geq \; h_{LB} = (\sum_{j=1}^{i} C_j - i)k + (i-1)(k+c) \tag{4.1}$$

Cost of Greedy Algorithm We implement the greedy algorithm to show that its cost is identical to the lower bound. Begin the greedy algorithm at any node in Cluster 1. The C_1-1 nodes within Cluster 1 are connected by a path in which the cost of each arc in the path is k. Once all Cluster 1 nodes are exhausted, the next node must be from another cluster, Cluster 2. To get to this node costs $k+c$. The total cost to get to a node in this next cluster is therefore $(C_1 - 1)k + k + c$ or $C_1 k + c$. In a similar manner, all nodes in Cluster 2 are then exhausted. The path then proceeds to the first node in Cluster 3 at cost of $C_2 k + c$ for going from the

first node in the sequence in Cluster 2 to the first node in Cluster 3 in the sequence. This continues until Cluster i is exhausted and the sequence is completed. The total cost excludes the cost of exiting the last cluster $(k + c)$. Therefore, the total cost obtained using the greedy algorithm (h_{GA}) equals:

$$h_{GA} = \sum_{j=1}^{i}(C_j k + c) - (k + c) \tag{4.2}$$

Finally, (4.1) and (4.2) are equal, proving that the greedy algorithm is optimal. QED.

Note that Theorem 1 does not require that the path begin at any particular node. Therefore, using a similar argument, we have the following corollary:

Corollary 1 *The value h associated with the shortest path for a TLHS is independent of the initial node.*

The sequence generated by the greedy algorithm has a second important quality. Any subsequence of an optimal sequence can itself be generated by employing the greedy algorithm on the subset of nodes included in the subsequence. Therefore, we obtain a second corollary:

Corollary 2 *Once the sequence of setups has been established by the optimal path, any subsequence of the sequence is also optimal.*

For example, if the optimal setup sequence is (v^1, v^2, v^3, v^4), then (v^1, v^3) is also optimal.

Imagine now that a number of $TLSH$'s exist and that the cost associated with all arcs between nodes of different $TLSH$'s is $k+c+c_1$ where $c_1 \geq 0$. This forms a three-level hierarchy. The proof of Theorem 1 can be easily extended to show that the lowest cost sequence can again be found via a greedy algorithm. Repeating this argument for multiple level leads to the final corollary:

Corollary 3 *The shortest path (smallest h) for an n-level hierarchical system can be found using a greedy algorithm.*

Stochastic Extension Theorem 1 proves the optimality of the greedy algorithm for the $TLHS$ with deterministic arc costs. Suppose that arc costs are random, but that the expected values of the arc costs have a hierarchical structure. Since the expected value of a sum is the sum of the expected values, the above proof can also be used to show that the greedy algorithm provides a sequence with the minimum expected cost. Additionally, if the variance is proportional to the mean, the same sequence is also the minimum variance sequence given that arc costs are independent of each other.

4.5.2 Bounds

In actual practice, Type 1 and 2 arc costs will not all be identical. Assume that Type 1 arcs have a cost of approximately $k + c$ and Types 2 arcs have a cost of approximately k. For this hierarchy the distributions of the arc costs of the same type arc are similar but not the same. Again, assume we are dealing only with the $TLHS$. We define a cluster to be any set of nodes between which there are no Type 1 arcs. We would like to bound the difference between the optimal value of

h and the value obtained using our greedy algorithm. Let:

L_1 = The cost of the lowest cost arc between any two nodes of different clusters.

G_1 = The cost of the highest cost arc between any two nodes of different clusters.

L_2 = The cost of the lowest cost arc between any two nodes of the same cluster.

G_2 = The cost of the highest cost arc between any two nodes of the same cluster.

h^* = The true minimum path cost.

h_1 = A minimum lower bound on h.

h_2 = The maximum value for h using the greedy algorithm.

Assume by the definition of the hierarchy structure that $L_1 > G_2$. We obtain h_{LB} from (4.1) and h_{GA} from (4.2) to get:

$$h_1 \leq h_{LB} \leq h^* \leq h_{GA} \leq h_2 \tag{4.3}$$

By determining the difference between h_1 and h_2, a bound is obtained on the difference between h_{LB} and h_{GA} which in turn bounds the difference between the greedy algorithm h_{GA} and the true optimal value h^*. We now find values for h_1 and h_2.

Minimum lower bound We use similar ideas as for the derivation of (4.1). There must at least be (i - 1) Type 1 arcs with a minimum cost of L_1 each, for a lower bound on the cost of all Type 1 arcs of (i-1)(L_1). Similarly for Type 2 arcs, there must be at least $\sum_{j=1}^{i} C_j - i$ arcs at a minimum cost of L_2 each, for a cost

of all Type 2 arcs of $(\sum_{j=1}^{i} C_j - i)L_2$. Consequently:

$$h_1 = (i-1)L_1 + (\sum_{j=1}^{i} C_j - i)L_2 \tag{4.4}$$

h_1 is less than h_{LB} because in calculating h_1, all arc costs take on their lowest possible cost while in calculating h_{LB} they do not. For example, if a $TLHS$ has 5 clusters, h_1 accounts for the cost of the 4 arcs connecting these clusters as $4L_1$. h_{LB} accounts for the four arcs as the sum of the four lowest cost Type 1 arcs, where three of these arcs may be greater than L_1.

Maximum result from greedy algorithm We derive this result in similar manner to the way we develop (4.2). To exhaust the nodes in each cluster $j \neq i$ requires $(C_j - 1)$ Type 2 arcs and one Type 1 arc. The exception is the ith cluster, which does not have to be exited and thus, does not require a Type 1 arc. The maximum cost of each Type 1 arc is G_1, for a total cost of all Type 1 arcs of $(i$ - $1)G_1$. The maximum cost of each of the Type 2 arcs is G_2, for a total cost of all type 2 arcs of $(\sum_{j=1}^{i} C_j - i)G_2$. Consequently:

$$h_2 = (i-1)(G_1) + (\sum_{j=1}^{i} C_j - i)G_2 \tag{4.5}$$

h_2 is greater than h_{GA} for a reason similar to that provided above for h_1 being less than h_{LB}.

Using (4.4) and (4.5) the maximum amount of error possible using the greedy algorithm can be bounded by:

$$\text{Maximum Error} \leq h_2 - h_1 = (i-1)(G_1 - L_1) + (\sum_{j=1}^{i} C_j - i)(G_2 - L_2) \tag{4.6}$$

4.5.3 Formal Algorithm

If we now assume that the nodes discussed within the traveling salesman problem are the machine configurations and the arc lengths are the setup times, we can employ a greedy algorithm for establishing the $\mathcal{L}$ List. The algorithm would simply choose as the next machine configuration, that setup which requires the least amount of time and which is not already in the sequence.

Function Descriptions

We formalize the concept of the dominant attribute by creating the following function which returns the level number of the dominant attribute associated with any pair of part types:

$$f(P_{a_1a_2a_3a_4}, P_{a'_1a'_2a'_3a'_4}) = \min(i : a_i \neq a'_i, i \in (1, 2, \cdots, N)) \tag{4.7}$$

For example:

$$f(P_{1,2,3,4}, P_{1,2,4,4}) = 3;\ f(P_{3,2,3,4}, P_{1,2,4,4}) = 1.$$

Function S_A^I determines a subset of all machine configurations which, when switching from Part Type A, have an I Level dominant attribute:

$$S_A^I = \{B : f(A, B) = I \text{ and } B \in Q\}. \tag{4.8}$$

That is:

$$\text{If } A = P_{a_1a_2a_3a_4},\ S_A^I \text{ is the set of all } P_{a_1\ldots a_{I-1}a'_I\ldots a'_4}.$$

The Algorithm

We now formally describe the algorithm. The $\mathcal{L}$ List is created one part type at a time through a recursive algorithm. Let P^n be the n'th part type added to the list and let Q^n be the set of all part types not yet added after P^{n-1}. The algorithm is as follows:

1. P^1 is the last part type produced on the system prior to running the algorithm. Let n=2.
2. $I^n = \max_i(f(P^{n-1}, P_i))$, for $P_i \in Q^n$.
3. Choose P^n to be any member of $S^{I^n}_{P^{n-1}}$.
4. Increment n. Let $Q^n = Q^{n-1}/P^{n-1}$. (Note: Q^n is the subset of Q^{n-1} which is not in P^{n-1}.) If Q^n is not empty, go to Step 2. Otherwise, stop.

In Step 2, we find the lowest level I^n (the level of lowest dominant attribute) associated with all the part types not previously sequenced. In Step 3, we determine the set of part types not previously sequenced requiring a Level I^n setup and choose one to be next in the sequence. In Step 4, the part type most recently added to the sequence is removed from a list of eligible part types Q^n and then the cycle repeats until all the part types are exhausted. As long as this sequence is maintained, the minimum setup time is achieved because of the hierarchical structure and partial sequence independence assumptions.

4.6 Algorithm Attributes

This type of algorithm has many advantages including: ease of data collection, ease of calculation, robustness and real time capability.

Ease of Data Collection and Storage

For a successful implementation of this algorithm, data must be gathered and updated on a regular basis. This would involve the collection of the probability distributions for the setup times between each pair machine configurations. Integer programming models used to normally solve such problems would require storing and updating all sequence dependent setup times (*e.g.*, a model of a system producing 700 part types would require 700 times 700 or 490,000 pieces of data, each requiring an associated piece of data requiring storage and updating).

An examination of the final algorithm (Section 4.5.3) indicates that if part types are classified by the attribute structure, very little setup data is required. One merely picks the next part not previously produced with the lowest level dominant attribute which can be encoded in an n dimensional vector ($n = 4$ on our bottleneck). One only needs to store each part type and its associated machine configuration (*e.g.*, if there were 700 part types with 4 attributes, only 2800 pieces of data are required.).

Ease of Calculation

With the above data in hand, the $\mathcal{L}$ List can be calculated quickly with a sorting routine on a standard database package. The logic is simple and can be understood by the factory personnel and management alike. In fact, a preliminary prototype for controlling one machine of the factory was programmed in less than 5 weeks with commercial software.

Robustness

The algorithm is robust in that it is not crucial that setup distributions be identical to get reasonable results from the greedy algorithms. The algorithm is also robust in that any sublist of the original list is also efficient. This is important for real-time lot sizing. For example, as the week progresses, the schedule may vary such that the scheduler is not certain at the beginning of the week whether a particular part type will be made that week. The scheduler must normally decide between minimizing setup time with or without the extra part type(s). With the setup hierarchy, only one sequence containing all part types must be calculated. Any possible combination of part types can then be represented as an optimal subsequence (Corollary 2).

Real Time Capability

The algorithm is easily implementable in real time because it need only be calculated once at the beginning of the week and then remains optimal for the remainder of the week. In addition, it would only take a few minutes to run mid-week, if required for any reason. However, the real-time component of the scheduling process focuses on lot-sizing (Chapter 5), which takes the results of the sequencing algorithm as given, allowing an overall real-time scheduling capability.

4.7 Summary

In this chapter, the setup sequencing component of a real-time factory scheduler is described to fit into the hierarchy of control created in Section 3.4. The scheduler is provided with a matrix of sequence dependent expected setup change

times. From this list an optimal sequence (with respect to minimizing total setup time for a given week) is obtained.

The problem of finding this sequence is formulated as a stochastic traveling salesman problem for which an exact solution is too difficult to obtain in general. A setup hierarchy is established. The idea of this hierarchy was generated from observations of the actual setup times within a manufacturing facility and previous work characterizing major and minor setups. By establishing this hierarchy, the setup sequence is generated by a greedy algorithm. While the method will not necessarily work for all setup structures, a bounding procedure was presented as a starting point for determining whether or not the algorithm is useful in specific situations.

Important attributes of the algorithm are that it is robust, easy to update and simple to communicate to factory personnel. With the resulting setup sequence or $\mathcal{L}$ List, the scheduler then must determine exactly how much of each part type in the list to make. In the next chapter, an algorithm for the real-time optimization of lot sizes is developed.

Chapter 5

Real-Time Lot-Sizing Policy

In Section 3.4.1, a two-level process is described for scheduling a manufacturing facility. Level 1 creates a setup sequence for the week (the $\mathcal{L}$ List of Chapter 4). This list is an input (Figure 3-6) to the real-time lot-sizing algorithm now developed. Our factory observations indicated that randomness must be accounted by the scheduler. However, the model in this chapter only accounts for the most significant source of randomness (setup times). In Chapter 7, the algorithm used in the factory prototype extends this model to account for failures and other randomness. This extension is based on the analysis of the solution to this model.

The first part type in the $\mathcal{L}$ List is made in some quantity yet to be determined. The $\mathcal{L}$ List contains more part types than might possibly be produced in a given week. Therefore, there may be part types in the list for which it is optimal to produce nothing (zero lot size). After completing production of this first part type, the machine sets up for the next part type in the list for which the algorithm dictates a non-zero lot size. This is then followed by the production of this part type. This continues until all the part types in the $\mathcal{L}$ List are exhausted.

This chapter concentrates on the second scheduling level: determining in real time (Section 3.1) how big to make each lot (*i.e.*, how much of each part type to produce). A review of the lot-sizing literature is presented in Section 5.1. A description of the specific problem is described (Section 5.2). This problem is then formulated as a dynamic program (Section 5.3). An example is proposed and tested via numerical experiments. From the characteristics of the optimal control policy, a closed-loop control heuristic is proposed (Section 5.4). Finally, the chapter concludes with summary remarks about the algorithm (Section 5.5).

5.1 Lot-Sizing Literature

Dispatching is the loading of parts into a system. Dispatching policies determine exactly how much and at exactly what time to load each part (Melnyk and Carter, 1986). *Lot sizing* focuses strictly on how much of each part type to produce. Many of the traditional approaches presented in the literature first calculate lot sizes and then sequence based on these lot sizes.

Dynamic lot sizing is often thought of as a variation of the Economic Lot Sizing Problem (Elmaghraby, 1978; Wagner and Whitin, 1958). Lot sizes are adjusted based on weekly demand and capacity fluctuations. The approach overlooks the details required for responding to short-term, random events. By contrast, we develop a short-term policy aimed at optimizing production in response to events (specifically, setup time variations) as they occur. In addition, setup times are also dealt with explicitly at a higher level (Chapter 4). Others approaches to the problem include deterministic optimization (Trigeiro, 1989) and queueing methods (Shioyama and Kise, 1989). However, these methods are frequently restrictive, tend to be more descriptive rather than prescriptive and are not practical in real time.

Surveys There are many surveys on lot-sizing. Zoller and Robrade (1988) summarize heuristic methods for dynamic lot sizing. In addition, many reviews of production scheduling contain sections on this topic (Graves, 1981; Karni, 1986; Fleischmann, 1990; and Bahl, Ritzman and Gupta, 1987).

5.2 Problem Description

Our system is one machine with significant and random setup times. Every Monday morning, orders for a number of different part types are provided whose production must be scheduled for delivery by Friday evening of that same week. The number of hours available for production is fixed. Orders required by the end of the current week are frozen (Sridharan and Berry, 1990) and may not be altered. Projections for the following week are provided at the same time. These projections rarely differ from the actual orders. We assume all the conditions discussed in Section 4.4.1 remain valid and that the sequence in which part types are produced is given by the algorithm of Chapter 4.

Our system is only available a limited amount of time each week for production and setups. To obtain the actual working time (T), scheduled downtime (*e.g.*, lunch, meetings, maintenance, etc.) is subtracted from the time between demand arrival and finished goods shipment (current week due date). (Overtime is available, but at a high cost. Therefore, available machine time is assumed to be fixed.) Keeping in mind that we are not accounting for failures in our model, the following relation is presented:

$$\begin{aligned} T &= \text{the amount of time available each week for production and setups} \\ &= \text{(Current due date - Arrival time of demand) - (Scheduled downtime)} \end{aligned} \tag{5.1}$$

For a normal week of five working days, each with 20 hours of potential machine up time, a maximum of 100 hours of production hours are available. If 4 hours are required for scheduled maintenance and 6 hours are required for training, T would equal 100 minus 10 or 90 hours.

Time for downstream processing must also be accounted for to assure delivery by the Friday due date. In the system description, all downstream processing times are assumed to be negligible with the exception of Stage Two. The total downstream processing time can be accounted for by subtracting from T the time required for the last part processed at the bottleneck required in the current week, to get through the rest of the system with a high level of confidence. This amount of time should be treated identically to scheduled downtime. However, it can be incorporated into the new scheduling cycle of the following week.

Management is faced with conflicting interests. Meeting current requirements with a high level of confidence is very important. However, by producing larger lot sizes, the probability that all current requirements will be completed within T is reduced. For example, assume five part types are required in the current week. Spending more time producing the first part type results in less time being available for future part type production. Therefore, if the first lot is too large, there will not be enough time to complete the rest of the current requirements by the end of the current week. However, there is an advantage to larger lots. By producing larger lots, fewer setups are required on average, and therefore more useful production may be achieved with less machinery. Lot sizes are required which establish an acceptable balance among the system capacity, the probability of completing current requirements and inventory costs in light of system uncertainty (*e.g.*, random setup times).

In our system, inventory has a very low cost, little obsolescence and requires little space. This low inventory cost along with the excess short term capacity in the long term system bottleneck (Stage One) suggests that a limited build-ahead policy is reasonable. This assumption opposes current thinking such as Just-in-Time (Golhar and Stamm, 1991) which advocates very low inventories. However, with excess capacity (Section 3.2.2) and low inventory costs, a logical question that could be asked is, "Why not just overbuild and keep very large inventories?" In reality, some very infrequently produced parts requiring only small yearly quantities will be made in advance. However, for the majority of parts produced, this will not be the case for a number of reasons. First, orders are only provided a limited number of weeks in advance and may vary substantially. Second, there are over 700 part types produced by the facility. Keeping large buffers of each of these part types would be a significant task and would also likely lead to obsolescence. Third, while the explicit cost of even large amounts of inventory is low, the logistics of tracking such inventory are considered undesirable by management. Therefore, for most part types, producing inventory is an acceptable policy as long as it limited strictly to what is included in the current demand projections.

Given inventories as an acceptable option, we now must determine how much inventory to produce. Thus, when the machine is operating, we must decide whether to continue building the current part type or to switch production to another part type. This decision must be made in light of the conflicting interests to complete current requirements and to free up future capacity by building larger lots. The success of the controller will measured according to the percentage of weeks during which current requirements are completed and the amount of setup time reduction is achieved by producing future requirements.

We make a number of additional assumptions to simplify the formulation of our model. These are:

- The setup sequence is fixed and all setup times have the same discrete prob-

ability distribution.

- Demands for all part types are identical.
- Demand projections are provided only one week into the future. In addition, we assume both weeks' requirements are identical.
- Failures are not treated.
- Production times are deterministic.

Most of these assumptions can be easily relaxed without affecting our ability to solve the problem and extend the model closer to real situations.

5.3 The Dynamic Program

The purpose of the control policy is to determine how much of each part type to make in real time as discussed in Section 5.2. It is assumed that an optimal sequence of part types (with respect to minimizing total setup time) is provided in the $\mathcal{L}$ List. The lot-sizing algorithm developed in this section determines the size of each lot in real time, while maintaining the part type sequence dictated by the $\mathcal{L}$ List. First, a brief description of the solution technique is provide (Section 5.3.1). This is followed by the model itself (Section 5.3.2).

5.3.1 Solution Technique – Dynamic Programming

Scheduling theorists frequently model manufacturing systems using unrealistic assumptions. These models require significant time and effort to solve, preventing their use in a real-time environment. For example, stochastic events are treated

deterministically by assuming that activities occur in accordance with their expected behavior. The purpose of our research is to develop a scheduling procedure which is implementable, theoretically sound and can handle randomness.

The *control action* is an action taken which will affect the future state of the system. A *control policy* determines how and when to employ the control action. *Closed loop feedback control* is a kind of control policy that specifies the control action based on the state of the system (Section 3.1). This type of control allows the stochastic nature of the system to be taken into account when scheduling. There are many possible ways of determining a closed loop control policy, including simulation, analytic models and artificial intelligence.

Determining a closed-loop policy for our factory lends itself to the dynamic programming technique (Bertsekas, 1987). A cost can be established as a function of the system state at the conclusion of the week. For example, this function can incorporate the combined costs of each current job not completed and the number of completed future jobs. The dynamics of the system can also be written as a set of equations. Numerical methods can then be used to determine a control policy which minimizes the cost.

5.3.2 The Model

In this section, we develop the dynamic program. We begin by observing that setups are the only random activities in our system and there is a finite number of them. Since decisions are only required upon the completion of random activities, a discrete dynamic programming model indexed to setup completions is formulated. For example, suppose a setup has just been completed (Figure 5-1, Decision Point i). Processing times are assumed to be deterministic. Therefore, the controller has as much information at the completion of Setup i as it does after

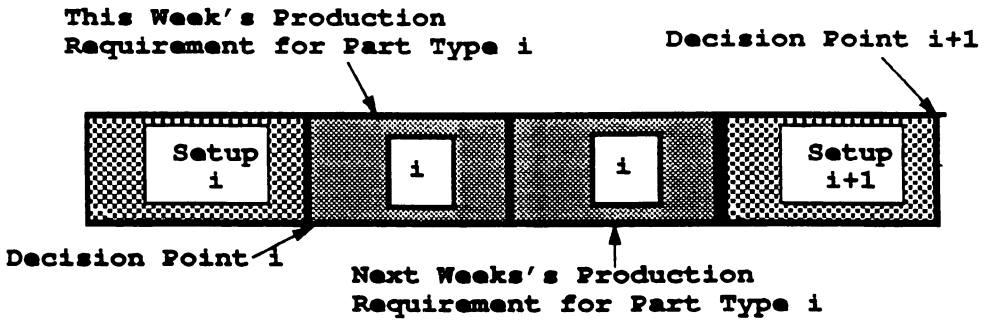

Figure 5-1: Decision Points.

the production of the current requirements of Part Type P^i. Setup $i+1$ is the next random activity. Upon the completion of each setup, the system may produce only the current requirements and immediately set up for another part type; or produce the next week's requirements in addition to the current requirements and then change setups (Figure 5-1). We now model this as a dynamic program.

System Parameters

T	=	The amount of time available each week for production and setups.
N	=	The maximum number of setups required within in a week.
i	=	An index indicating a part type's position in the $\mathcal{L}$ List.
D^i_m	=	Production time for current demand (D^i_1) or for projected demand of the following week (D^i_2) for Part Type i (m = 1,2).
S^i	=	Setup time i. This is a random variable which can take on two values which we call *Long Setup* and *Short Setup*.
$P(S)$	=	Probability mass function for S.
P_1, P_2	=	Penalty coefficients for not meeting current production.

System State

The system state contains all the relevant information for deciding whether to switch machine configurations or continue production of the same part type. The state x^i, upon the completion of a setup i, consists of current time (T_i) and the number of future lots completed (R_i):

$$\begin{aligned} T_i &= \text{The time of the } i\text{th setup completion.} \\ R_i &= \text{The number of future lots completed at } T_i. \\ x_i &= (T_i, R_i) \end{aligned}$$

Control Function

The control function is a mapping from the system state to a control action. The control actions are to build inventory or to switch setups:

$$\mu_i(x_i) = \begin{cases} 1 & \text{Build inventory with the current machine configuration.} \\ 0 & \text{Switch setups.} \end{cases}$$

Objective Function

The goal of the dynamic program is to develop a control policy which results in a good system performance. System performance can be measured as a function of the number of future and current requirements completed. We are only concerned with complete jobs because incomplete jobs do not eliminate any future setup requirements. The objective function is designed to reward a good system performance by increasing linearly with the number of future lots completed (R_i), and decreasing quadratically with the number of current jobs not completed by the end of the week (I^*). We would like I^* to be zero. A discussion on the choice

of P_1 and P_2 coefficients is presented in Section 5.4.2. The objective function is:

$$g_i(x_i) = \begin{cases} R_i - P_1 I^* - P_2(I^*)^2 & \text{if } T_{i+1} \geq T \text{ and } T_i \leq T \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

where

$$I^* = \begin{cases} N - i & \text{if } T\text{-}T_i \geq D_1^i \\ N + 1 - i & \text{if } T\text{-}T_i < D_1^i \end{cases} \tag{5.3}$$

T_i indicates the point in time when setup i is complete. The *if* condition in (5.2) indicates that at some point between the completion of setup i and setup $i+1$, production time ran out. Therefore, Part i is the last possible part produced in the current week. Condition (5.3) determines whether there is actually enough time (D_1^i) to produce Part i. In other words, once setup i has been completed, there are D time units are still required to complete job i. The two equations establish the terminal system state and allow the cost to be determined.

System Dynamics

The random variable t_i represents the amount of time between decision points i and $i+1$ (Figure 5-1). If the control action is to build an extra job, t_i is the time to produce the present requirement (D_1^i) plus the time required to produce a second job (D_2^i) plus the random length of the setup time (S). If the control action is not to build an extra job, t_i is the time to produce only the present requirement (D_1^i) and the length of the random setup time (S). Consequently:

$$T_{i+1} - T_i = t_i(x_i, \mu_i(x_i)) = D_1^i + \mu_i(x_i)D_2^i + S^i \text{ w.p. } P(S^i) \tag{5.4}$$

where S^i can be a *short setup* or a *long setup*.

The dynamics of accumulated future setup reductions is governed by:

$$R_{i+1} - R_i = \mu_i$$

Then:

$$\begin{aligned} x_{i+1} &= (T_{i+1}, R_{i+1}) = f_i(x_i, \mu_i, t_i) = (T_i + t_i, R_i + \mu_i) = x_i + (t_i, \mu_i) \\ &= x_i + (D_1^i + S_1^i, 0) + \mu_i(x_i)(D_2^i, 1) \end{aligned} \tag{5.5}$$

General Program Formulation

Since the objective function has a non-zero value only at the end of the week (5.2), the cost-to-go function (J) at T_i is written as an expectation (E_i) of the objective function value at the last decision point i^* of the week. That is:

<u>Cost-to-Go Function</u>

$$J_i^*(T_i, R_i) = \max_{\mu_i, \ldots, \mu_{i^*}} E_i \left[g_{i^*}(T_{i^*}, R_{i^*}) \right] \text{ s.t. } T_{i^*+1} \geq T \text{ and } T_{i^*} \leq T \text{ and } (T_i, R_i) \text{ satisfy (5.5).} \tag{5.6}$$

Recursive Dynamic Programming Equations

We generate the solution to (5.6) by recursively solving the following equations:

$$\begin{aligned} J_i^*(T, R) &= \max_{\mu_i} E \left[J_{i+1}^*(f_i(T_i, R_i, \mu_i, t_i)) \right], \text{ for } i = 1 \text{ to } n \\ &= \max_{\mu_i} E \left[J_{i+1}^*(T_i + D_1^i + \mu_i D_2^i + S^i, R_i + \mu_i) \right] \end{aligned} \tag{5.7}$$

5.4 An Example

An example is presented to demonstrate how an optimal controller might behave (Table 5.1). Ten part types require processing in the current week and there is a projection for the same amount in the following week. Each part type requires exactly 5 units of processing time each week. There is 100 time units available on the bottleneck machine. Setup times are significant and require 5 time units with probability 0.5 or 2.5 time units with equal probability. Different values for the penalty coefficients P_1 and P_2 were tested.

$$
\begin{array}{lcl}
T & = & 100 \\
N & = & 10 \\
D_m^i & = & 5 \text{ for } m = 1,2 \; i = 1 \text{ to } 10 \\
P(S^i) & = & \begin{cases} 0.5 & \text{for } S^i = 2.5 \\ 0.5 & \text{for } S^i = 5 \\ 0 & \text{otherwise.} \end{cases}
\end{array}
$$

Table 5.1: Example Parameters

In this section, we begin with a description of a numerical derivation of the control policy (Section 5.4.1). The control policy is observed to have a threshold type nature. A couple of other thresholds are then presented (Section 5.4.2). Finally, we conclude with a suggested heuristic (Section 5.4.3).

5.4.1 Numerical Derivation of Control Policy

An analytical expression has not been found for the cost-to-go function (5.6). However, the parameters from Table 5.1 were used in conjunction with (5.7) to generate a computer program written in the C language to numerically generate the control policies. Using this program, experiments were conducted with various

P_1 and P_2 values to determine the control action associated with each system state. Figure 5-2 indicates the control policy associated with each sample point when P_1 and P_2 are both 1.1. This picture is indicative of the type of partitioned sample space associated with many different P_1 and P_2 values. Variations in R (the number of completed future jobs) had no effect on the control policy. The linear structure of the cost associated with the number of future completed jobs provides an intuitive reason for this indifference. In other words, the change in the cost function due to the current chosen action is independent of any past reward. Therefore, R is not accounted for when partitioning the state space in Figure 5-2.

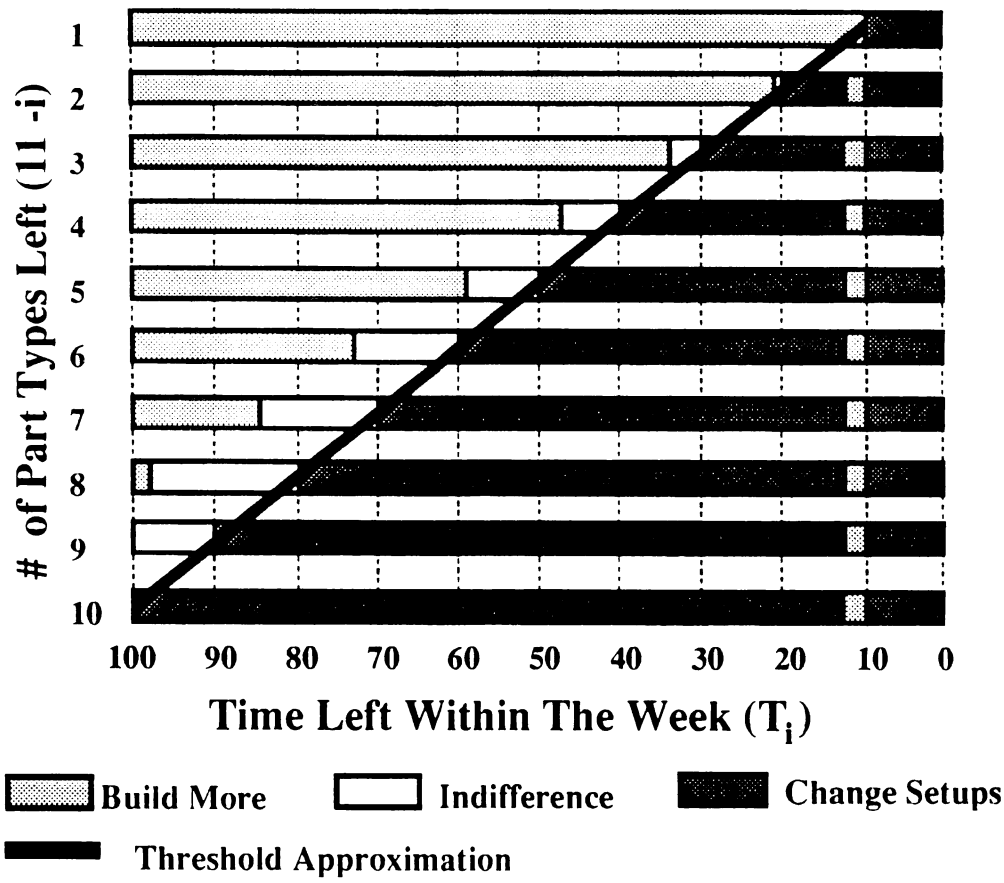

Figure 5-2: Control Policy

Different shadings of the remaining state space represent different control actions associated with each possible system state (*i.e.*, light grey represents a *build more* action, dark grey a *switch setup* action and white is an *indifference region* between the two actions).

Indifference Region

The value of the cost-to-go function is insensitive to the control action for system states in the *indifference region* (Figure 5-2). Although no formal proof of a reason for the existence of this region is provided, we suggest that it is caused by the demand structure. More specifically, since the benefit associated with completing all extra jobs is the same, the controller is indifferent to which extra jobs are completed, for a limited time horizon; the cost function only accounts for the total number of extra jobs it expects to complete in the future. For all points to the right of the right boundary of the indifference region, the risk associated with not completing current requirements exceeds the benefit of completing an extra job. For all points to the left of the left boundary, the number of current part types yet to be produced falls below a threshold, forcing the loss of an opportunity to make at least one extra job. For example, suppose the controller calculated that five extra jobs could be completed. Once there are only four jobs left in the $\mathcal{L}$ List, the opportunity to make the fifth extra job is lost. Therefore, as long as no opportunities are lost and current requirements are not threatened with a given level of confidence, the cost function is insensitive or indifferent to the control action.

In implementing our numerical simulation in Chapter 6, we choose to assume an aggressive policy is employed, when possible. Therefore, the *build more* option is the control action for all points of indifference. We believe that this indifference region will provide insight into developing more detailed policies for jobs of varying degrees of importance in possible future research.

End Effects

Figure 5-2 also indicates to always *build more* when T_i is between 10 and 12. The reason for this is that the model determines that not enough time is available to set up and produce the next part type after completing the current job. To maximize the cost-to-go function, the controller determines that the negative cost associated with not completing the next currently required job must be incurred. Therefore, to maximize the expected value of the cost function, the controller tries to build extra inventory. However, all sample points representing more than one remaining job are strictly theoretical. These points represent situations when there is significantly less time left than is required to complete the remaining jobs. Our assumptions about capacity and conservativeness would prevent this from ever happening. Therefore we overlook this anomaly in the partitioned sample space.

Threshold Approximation

By assuming we build *build more* in the indifference region and that the *switch setups* action in the anomalous region is unimportant, a threshold policy represented by the *threshold approximation* (Figure 5-2) is suggested. For every number of jobs (11-i) left in the system, there is an associated time α_i. If more time than α_i is left in the week, build inventory; otherwise, do not. With an efficient method of determining this threshold, an implementation would be trivial.

5.4.2 Other Thresholds

In actual practice, business conditions may vary from week to week (*e.g.* anticipated future capacity shortages, lack of available overtime, inventory shortages, etc.). The variability associated with these conditions is managed at a higher level

than our controller. However, these changes have the effect of altering P_1 and P_2 in an objective function. For example, if backorders were very high, the emphasis on completing current requirements might be replaced by a large lot policy which would be represented by lower P_1 and P_2 values. Therefore for each set of P_1 and P_2 there is an optimal policy which we approximate by an *optimal threshold approximation.* In the limiting case as P_1 and P_2 grow very large, we obtain as the optimal policy what we call the *conservative threshold.*

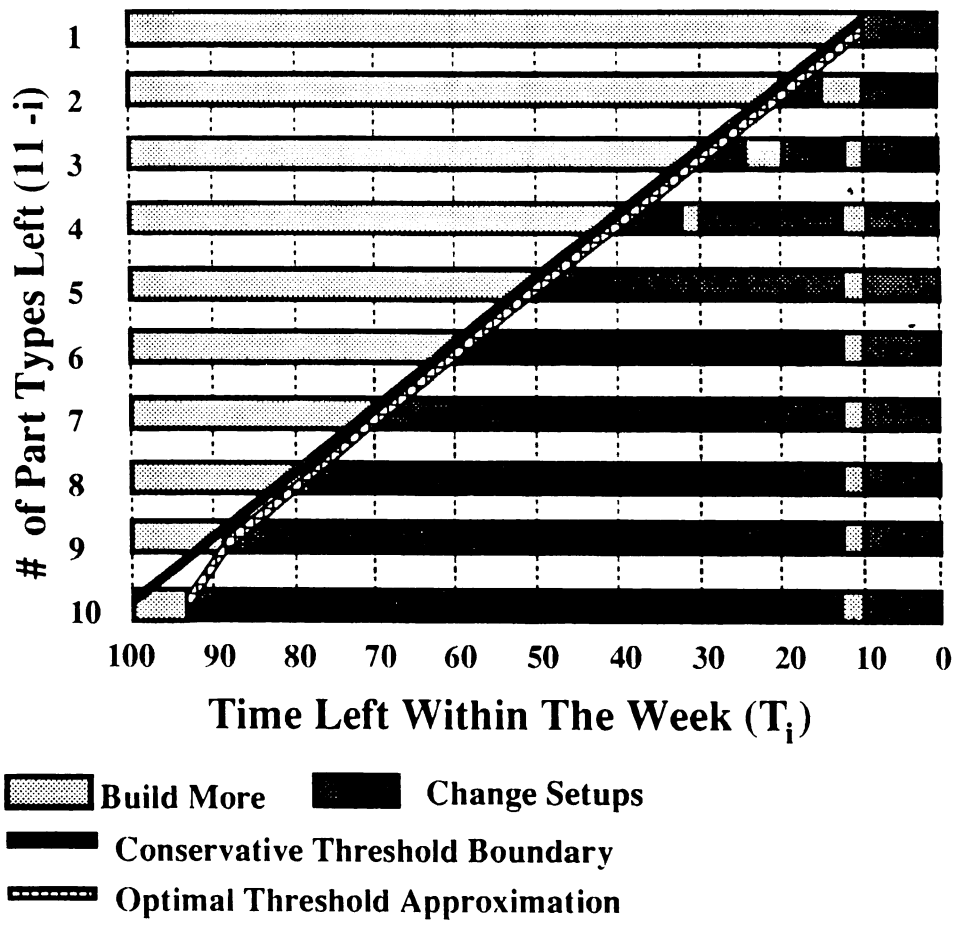

Figure 5-3: Other Thresholds

Conservative Threshold Boundary

When business conditions are such that meeting current requirements takes precedence over building any inventory, a conservative strategy is required. The *conservative threshold* illustrated in Figure 5-3 is the leftmost feasible threshold. Notice that in this and all other illustrations the *indifference* region has been incorporated in the *build more* region. It is called conservative because it assumes the worst-case scenario (for our example, all future setup times require five time units, the largest value of S).

To produce extra inventory when the system is in a state to the left of this line could never jeopardize current production. In a sense, this threshold is a minimum bound on all thresholds. No matter how great are P_1 and P_2, the threshold can never be any further to the left. This assertion was tested and verified via numerical experiment. Our assumptions about discrete and bounded S^i values make this an easy threshold to determine. One need only determine the maximum amount of time associated with each future activity. For example, if Setup 3 was just completed at T_3 equal to 20, the system must complete 7 more setups (7 times 5 or 35 time units) and 8 more jobs (8 times 5 or 40 time units) within 80 time units (100 minus 20). In the worst case, there is still 5 time units left over to build inventory. Therefore, future requirements for 1 job can be produced without ever jeopardizing the current requirements.

In reality, S^i may not be bounded and will definitely take on continuous values. Therefore, it is impossible to allocate the maximum amount of time to each random activity. An extension to this model which allows for unbounded continuous values of random variables in the manufacturing system is described in our implementation description (Chapter 7).

Optimal Threshold Approximation

We assume that P_1 and P_2 exist such that an *Optimal Threshold Approximation* could be found to meet different desired objectives. In the case illustrated in Figure 5-2 when P_1 and P_2 are set to 1.1, the optimal threshold approximation is, in fact, equivalent to the conservative threshold. Intuitively, an implementation of a policy with P_1 and P_2 values greater than 1.1 is equivalent to a more conservative policy. This would imply that the threshold curve would move up and to the left. However, since the conservative boundary is a limiting threshold, for all values of P_1 and P_2 greater than 1.1, the partitioned sample space would indicate to use the conservative threshold. This was verified by numerically testing many (P_1, P_2) values greater than 1.1.

In reality, our factory sets an arbitrary limit of completing current requirements at least 95% of the time while maximizing the number of completed future jobs. While we did determine that the (P_1, P_2) values associated with meeting this objective could not both be greater than 1.1, the number of other possibilities is infinite because P_1 and P_2 are real. Therefore generating all thresholds for all possible (P_1, P_2) values is an impossible task. However, after testing many values with the simulation method described in Chapter 7, the policy (which arbitrarily assumes that the control action associated with points in the indifference region is *to build*) represented in Figure 5-3 maximizes the number of extra jobs completed while completing current requirements at least 95% of the time. The associated (P_1, P_2) values were both set at 0.877. This policy was approximated by a threshold which is labeled *Optimal Threshold Approximation* in Figure 5-3. Experiments also indicated that other closely related (P_1, P_2) pairs resulted in similar results.

5.4.3 The Optimal Threshold and the Heuristic

In the conservative environment of this and most factories, we noticed that the *optimal threshold approximation* starts close to, but below the *conservative threshold approximation* and approaches it as the week progresses (Figure 5-3). The conservative boundary is easy to calculate as described above. Therefore, we thought the conservative boundary might provide a reasonable heuristic threshold which could be used to generate a control policy. In the next chapter we compare the results of using such a threshold via numerical experiments.

5.5 Summary

In this chapter, a real-time lot-sizing scheduler is developed which makes use of a pre-specified ordering of part types (the $\mathcal{L}$ List) scheduled for production in the current week. The scheduler is given this ordering and must decide how much of each part type to produce. Feedback control is suggested as a methodology for modeling, controlling and analyzing the problem.

The problem is formulated as a dynamic program. While the program is formulated in a general context, no closed form solution is presented. A simplified example is presented for numerical experimentation. Upon examining the resulting partitioned sample space from the experiments, the optimal policy appeared to be of a threshold nature. Using a linear approximation to this policy, a heuristic is generated. Varying the cost functions with different penalty values indicated the robustness of this policy. The partitioned sample space of almost all the (P_1, P_2) values tested resulted in approximate threshold policies. In the next chapter, our heuristic threshold feedback policy is compared with more traditional approaches in the facility via numerical experiments.

Chapter 6

Simulation Experiments

The previous five chapters develop a methodology for scheduling production via real-time, feedback control. The analysis of the resulting partitioned state space suggests the use of a simple threshold policy. However, this policy alone provides no indication of what the resulting events would be if the policy were actually implemented (*e.g.*, percentage of weeks in which current requirements are completed, the expected number of extra jobs completed, etc.). Simulation experiments are required to determine the effects of using different policies.

This chapter describes these experiments, beginning with their design (Section 6.1). Characteristics of the simulated open-loop policies are provided (Section 6.2). This is followed by the set of simulated closed-loop policies (Section 6.3). The results of the simulations are then presented, indicating the relative performance of each controller (Section 6.4). The chapter concludes with brief summary comments (Section 6.5).

6.1 Simulation Test Design

Simulation is a method used to test the effectiveness of a given control policy without an actual implementation. Our simulation generates a probabilistic sample sequence of random events (10 setup durations). The different control policies are applied to the simulated environment in which these sequences of events would occur. The outcomes of the simulations are then used to deduce the behavior of the policies.

For the example problem (Section 5.4), there are 2^{10} different feasible random event sequences (10 setup completions, where each setup has two possible durations). For most simulations, a set of sample paths are generated. However, we have the available computing power to test the performance of all controller for each of the 2^{10} possible sequences. The results of these experiments provides the exact distributions of the number of extra jobs completed in one week for each policy. From these distributions, one can calculate the expected number of extra jobs completed, the variance of this number as well as the percentage of weeks in which current requirements are not completed. Keep in mind, we are only simulating one week and not successive weeks.

6.2 Open-Loop Policies Used in Practice

Open-loop policies have no feedback of the system state in the selection of the control action. Based on informal observations, we found that scheduling policies used in practice are often open-loop policies. The schedule is determined once at the beginning of the week and then never modified. The four open-loop policies we found used in industry are: the idle-time-waster, the open-loop aggressive policy, the open-loop conservative policy and the open-loop expected value policy.

Idle-Time-Waster

We define the *idle-time-waster* as a policy in which excess time is allocated to account for system variability and inventory is not produced. The machine is idle when all of the time allocated to jobs is not required. With the advent of Just in Time (JIT) practices (Golhar and Stamm, 1991), many manufacturers accept idleness on machines in exchange for on-time delivery and low inventories. Production plans are constructed with large periods of potentially unused time.

This policy is reasonable when the cost of capacity is not an issue. However, this is not the case in most factories. When management uncovers idle machines, additional requirements are often demanded of the system to improve utilization rates of expensive capital equipment. In addition, when a great deal of excess capacity is available, scheduling is not a critical issue. Therefore, this case is not examined any further.

Open-Loop Aggressive Policy

Machine operators frequently realize that increasing lot sizes has a number of advantages. Fewer setups are required which results in increased capacity. To obtain the benefits of larger lot sizes, machine operators may establish an *open-loop aggressive policy* in which the control decision is to always build inventory. The control policy is simply:

$$\mu_i(x_i) \quad = \quad \left\{ \; 1 \quad \text{Build inventory, for } i = 1 \text{ to } 10. \right.$$

Unfortunately, the results of such a policy are that the present requirements are rarely completed unless current requirements have been completed in a previous or there are very few future requirements. This is unacceptable in the due date framework assumed in this paper. Therefore, this policy is not examined any

further.

Open-Loop Conservative Policy

Operating in opposition to the aggressive policy are the operators who never risk not completing the current requirements. An *open-loop conservative policy* dictates that no inventory is produced before current requirements. Consequently:

$$\mu_i(x_i) \quad = \quad \left\{ \; 0 \quad \text{Switch setups, for } i = 1 \text{ to } 10. \right.$$

If excess time exists upon the completion of current requirements, inventory may be produced. This is a common practice in factories. We make an additional assumption for our simulation that the first part type of inventory produced is the same as the last part type produced for the current requirements.

Open-Loop Expected Value Policy

An *open-loop expected value policy* accounts for the time of activities based on their expected durations. In our sample problem, the expected setup time is 3.75 time units. 10 setups require 37.5 time units and current production of 10 part types requires 50 time units. Average setup time and minimum production require a total of 87.5 time units, leaving 12.5 time units available for building inventory. This is enough time to build two extra jobs worth of inventory. We assume the scheduler produces extra inventory for the first two jobs in the $\mathcal{L}$ List. Consequently:

$$\mu_i(x_i) \quad = \quad \begin{cases} 1 & \text{Build inventory, for } i = 1 \text{ to } 2. \\ 0 & \text{Switch setups, for } i = 3 \text{ to } 10. \end{cases}$$

As with the open-loop conservative policy, any extra time left at the end of the week may be used for additional production, starting first with an extra job of the last currently required part type.

6.3 Closed-Loop Controllers

In addition to the optimal threshold approximation and heuristic policies discussed in Chapter 5, there are a number of other possible threshold policies which we analyze due to their ease of calculation and/or use in industry. These policies include a closed-loop conservative policy and the more popular closed-loop expected value policy. Both of these policies have the advantage that the threshold is linear and can be calculated very easily.

All thresholds policies are similar in that there is an α_i at associated with the completion of the ith setup. If $T - T_i$ (the amount of time left in a week) exceeds α_i one should build inventory and one should switch setups, if not. Consequently:

$$\mu_i(x_i) = \begin{cases} 1 & \text{Build inventory if } (T - T_i) \geq \alpha_i. \\ 0 & \text{Switch setups if } (T - T_i) < \alpha_i, \text{ for } i = 1 \text{ to } 10. \end{cases}$$

We now describe the closed-loop policies we simulated.

Optimal Threshold Approximation Policy

The factory must maximize the number of completed future jobs while meeting current requirements at least 95% of the time. This objective was best achieved when the dynamic program (5.6) was run with $P_1 = P_2 = 0.877$, resulting in the optimal threshold approximation (Figure 6-1). The closed-loop policy which

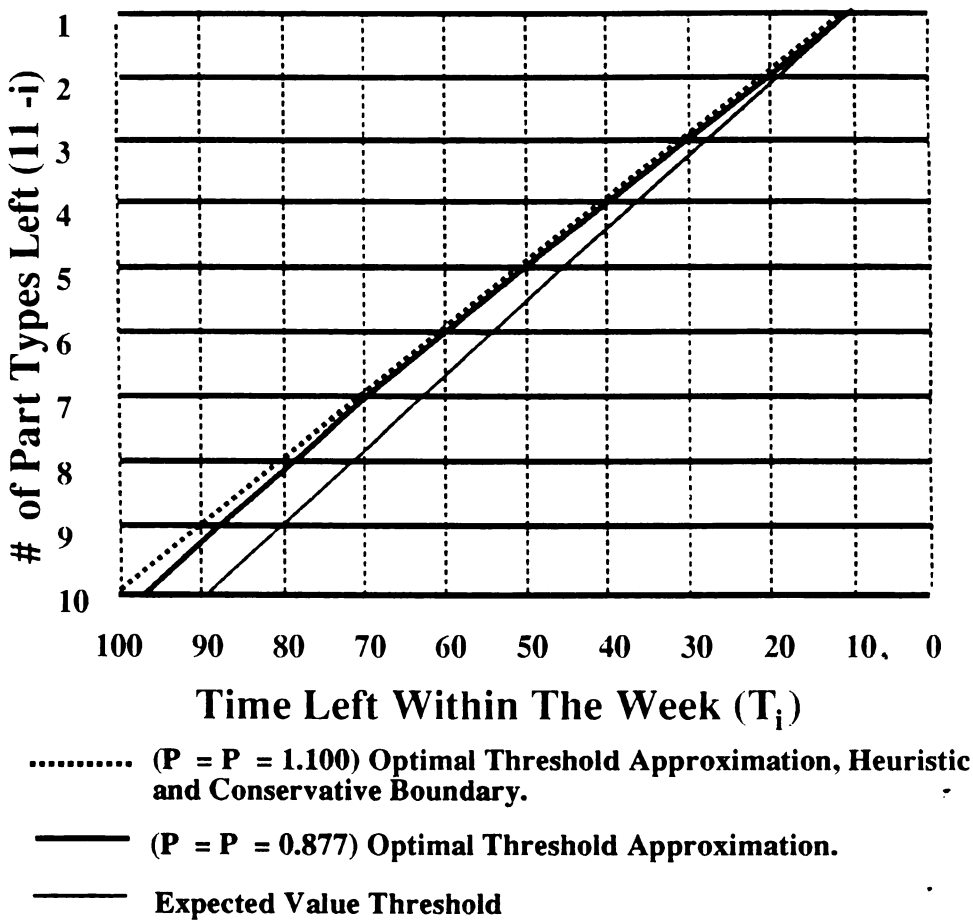

Figure 6-1: Closed-Loop Controller Thresholds

employs this threshold (α_i^{opt}) is henceforth called *optimal threshold approximation policy* or the optimal control policy.

Conservative Boundary Policy

The *conservative boundary policy* is a threshold policy (Figure 6-1) where the threshold is the conservative boundary described in Section 5.4.2. This threshold can be easily calculate by assuming that the $11 - i$ remaining jobs require 5 time units (D) each, the $10 - i$ remaining setups require 5 time units (S) each and the

extra job also requires 5 time units. Consequently:

$$\begin{aligned}\alpha_i^c &= \max_{S^i}[(11-i)D + (10-i)S^i + 5], \text{ for } i = 1 \text{ to } 10. \qquad (6.1)\\ &= (11-i)5 + (10-i)5 + 5\\ &= 110 - 10i.\end{aligned}$$

Heuristic Policy

Figure 6-1 indicates that the optimal threshold is very similar to the conservative boundary threshold. In Section 5.4.3 we suggest approximating the optimal threshold with the conservative boundary. Therefore, the *heuristic policy* is the same as the conservative threshold policy and uses the same threshold ($\alpha_i^h = \alpha_i^c$).

Closed-Loop Expected Value Policy

The *closed-loop expected value policy* is also a threshold policy, where the closed-loop expected value threshold (Figure 6-1) assumes that all future activities require their expected amount of time. By assuming that each of the $11 - i$ remaining jobs requires $D = 5$ time units, that each of $10 - i$ remaining setups requires $S = (0.5)5 + (0.5)2.5 = 3.75$ time units and that one extra job requires 5 time units, we obtain:

$$\begin{aligned}\alpha_i^{EV} &= E[(11-i)D + (10-i)S^i + 5], \text{ for } i = 1 \text{ to } 10. \qquad (6.2)\\ &= (11-i)5 + (10-i)3.75 + 5\\ &= 97.5 - 8.75i.\end{aligned}$$

Expected values are commonly used in many scheduling packages.

6.4 Test Results

Five control options were tested: a) the *optimal* threshold approximation policy with P values equal to 0.877, b) the closed-loop expected value policy, c) the *heuristic* policy (same as closed-loop conservative threshold policy and the optimal threshold approximation policy with $P_1 = P_2 = 1.1$), d) the open-loop conservative policy and e) the open-loop expected value policy. For each policy, the expected value and variance of the number of jobs completed is provided. In addition, the distributions of extra jobs completed in a week are also provided.

There are three important results from our tests (Table 6.1). First, the heuristic performed reasonably compared with the optimal policy. Second, it performed significantly better than its open-loop conservative counterpart. Finally, the performance of the expected value controllers was poor.

Heuristic vs. Optimal

Using the optimal policy current requirements were completed 96.1% of the time, which is within the system objectives (greater than 95%). The expected number of extra completed jobs were similar for both policies. The expected number of extra jobs was 2.32 for the optimal controller, which is not significantly better than the 2.25 for the heuristic. This difference is due to the overly conservative nature of the heuristic (which always completes current requirements) when compared to the optimal controller.

Heuristic vs. Open-Loop Conservative

The heuristic performed significantly better than its open-loop conservative counterpart in terms of the expected number of extra completed jobs (2.25 vs.

1.51). Both policies resulted in 100% of current requirement being completed. While the expected number of extra completed jobs for the heuristic was slightly worse than the open-loop expected value controller (2.25 vs. 2.38), this was more than made up for by its superior on-time performance (0.0 vs. 17.9 percent of the time current requirements were not completed). These figures were also based on the assumption that the controller had a requirement to build inventory for the last part type produced for current demand. The results would have been much more favorable to the heuristic, if a setup were required for the next part type after completing the current requirements.

Closed-Loop vs. Open-Loop

Both expected value schedulers had poor of on-time delivery results (31.6 and 17.9 percent of the time current requirements were not completed), without a substantial improvement in the expected number of extra jobs completed (2.58 and 2.38). The poor performance of these expected value controllers is stressed because of their prevalence in industrial applications and software used for scheduling. The results of these tests may begin to provide a picture of why these schedulers perform so poorly in practice in environments with random events.

6.5 Summary

The results of our tests clearly show that closed-loop control performs better than open-loop policies in an environment with random setup times. Figure 6-2 illustrates that the expected number of extra jobs completed using the heuristic was close to that of the best policies and Figure 6-3 illustrates that this performance was achieved with 100% on-time performance. This on-time performance record greatly exceeded the expected value controllers. We believe these results can be

duplicated in other manufacturing environments where variance is significant and there are other phenomena besides setups times.

Controller	% Inc. Weeks	% of Wks. completing x extra Jobs x = 0	1	2	3	4	5
Optimal ($P_1 = P_2 = 0.877$)	3.91	0.0	13.6	46.8	34.0	5.6	0.1
$E = 2.32, \sigma^2 = 0.6$							
Closed-Loop Expected Value	31.6	0.0	0.0	48.5	44.9	6.4	0.1
$E = 2.58, \sigma^2 = 0.4$							
Heuristic ($P_1 = P_2 = 1.1$) Closed-Loop Conservative	0.0	1.1	16.1	45.1	32.2	5.4	0.1
$E = 2.25, \sigma^2 = 0.7$							
Open-Loop Conservative	0.0	1.1	48.9	48.1	1.9	0.0	0.0
$E = 1.51, \sigma^2 = 0.3$							
Open-Loop Expected Value	17.9	0.0	0.0	62.3	37.1	0.6	0.0
$E = 2.38, \sigma^2 = 0.2$							

Table 6.1: Performance Summary

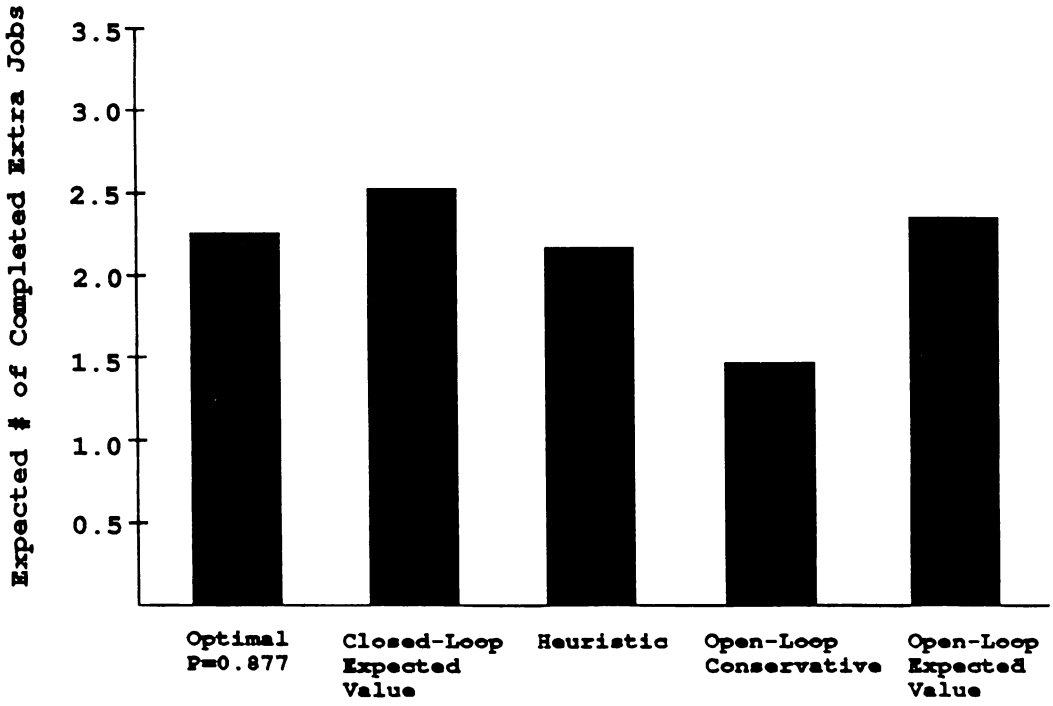

Figure 6-2: Expected Number of Extra Jobs Completed

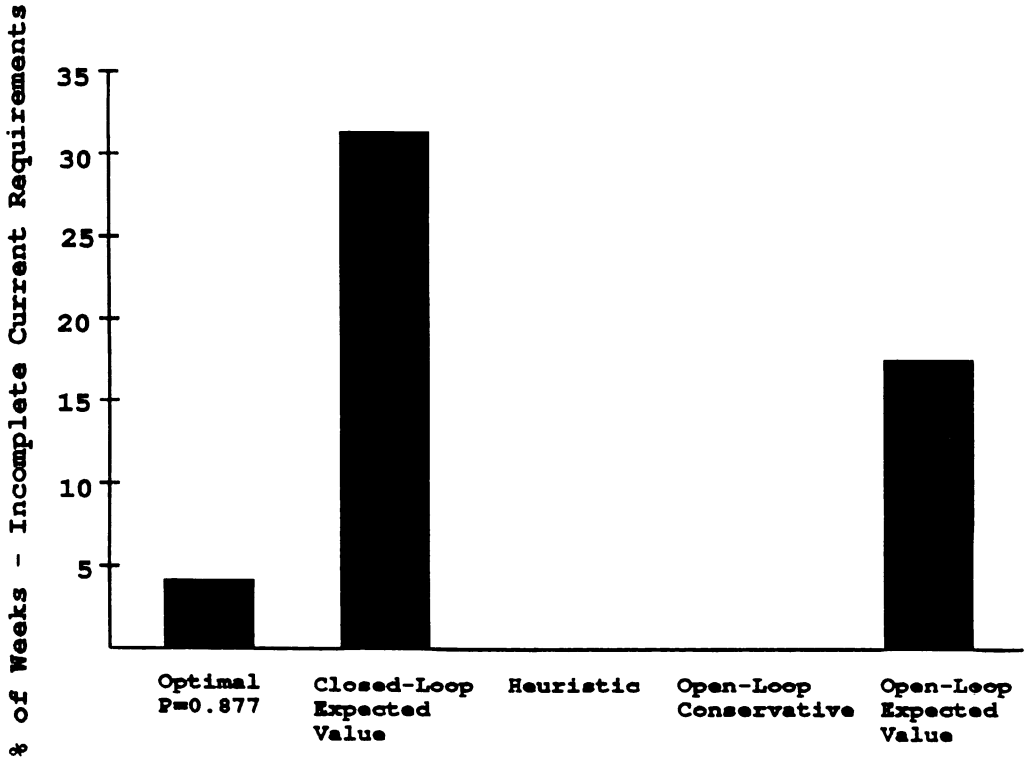

Figure 6-3: Percentage of Weeks with Incomplete Current Requirements

Chapter 7

Extensions and Implementation

A prototype software program was developed based on the results of our research. The prototype was designed for a machine only producing seven different part types and a number of extensions to the original models were required. Instead of using the hierarchy as a technique for simplifying sequencing, we exploited a similar but simpler ladder structure. In addition, we extended the lot-sizing heuristic to account for more randomness than just setup times. We make no attempt to prove the optimality of these extensions.

The implementation process began with isolating a section of the factory as an experimental testbed (Section 7.1). A number of modifications and theoretical conjectures are outlined which were required to accommodate the testbed conditions (Section 7.2). The software was developed (Section 7.3). Finally, future plans for on-line testing and research of broader adaptations were proposed (Section 7.4).

7.1 Project Testbed

A prototype software program was developed for an experimental testbed machine. Management isolated part of the factory to test the controller, because a full-scale implementation would be very expensive in terms of money, time and the opportunity loss, if unsuccessful. The machine used for this development work was similar, but not identical to the machine studied in the previous year.

Some important characteristics of the experiment machine were the same. Excess capacity was available up- and downstream so that the machine was a system bottleneck. Management agreed to only make demands of the machine which were feasible for a given level of confidence. Setup times were sequence dependent, significant (on the order of an hour) and random. Processing times were deterministic and identical for all part types. Setup time distributions were independent of each other. The processing rate was at least an order of magnitude greater than the setup rate. The setup rate was an order of magnitude greater than the rates of higher level decisions (*e.g.* capacity changes, process enhancements, etc.). A CIM system was in place for gathering all relevant data.

There were also some important differences. The machine would initially handle only seven part types. The setup time distributions did not form a hierarchy and had continuous distributions. In addition, setup times were unbounded. Failures were significant, had continuous unbounded density functions and could occur an unlimited number of times within one week. Machine yield was also important. In the next section, the necessary extensions to account for these differences are described.

7.2 Theoretical Extensions

As described in Section 7.1, the testbed had a number of differences from the machine originally studied which required attention. A method of accounting for the non-hierarchical setup structure is described in Section 7.2.1. Section 7.2.2 discusses the extensions to the lot-sizing algorithm (Chapter 5) which accommodates failures and yields, in addition to setup time variability.

7.2.1 Lot-Sequencing Extensions

As discussed in Chapter 4, determining a setup sequence for a machine with sequence-dependent setup times may be costly in terms of data storage, data maintenance and computer time. We show that special setup structures (hierarchies) may be exploited to substantially reduce the complexity of the sequencing problem. The setups for our testbed machine did not form a hierarchy, but they did have what we call a *ladder structure*. A ladder structure is defined as a setup structure in which the setup times are increasing with the difference in magnitude of a particular parameter associated with the machine configuration. For example, our test machine produces seven part types. The machine configurations required for each of these seven parts are identical with the exception of one parameter. When the this parameter is c^1 for producing one part and c^2 for a second part, the setup time to switch between these two is $f(|c^1 - c^2|)$ with $f > 0$. This ladder structure achieves the minimum total setup time by sequencing part types in the order of ascending or descending parameter value. Assume the name of any part type is the value of this key parameter. For the seven part types: 21, 65, 45, 32, 98, 34 and 15, the minimum setup time could be achieved with either:

$$(15, 21, 32, 34, 45, 65, 98) \text{ or } (98, 65, 45, 34, 32, 21, 15). \tag{7.1}$$

The minimum setup time could be obtained for any subset of these part types by selecting the appropriate subsequences of (7.1). For example, for the set of part types (15,98,34), the minimum time would be achieved with either the (15,34,98) or (98,34,15) sequence.

A similar procedure finds the minimum setup time sequence beginning with a specific part type. For example, if 32 was the part type for which the machine was configured, the optimal sequence would be among: (32,15,21,34,45,65,98), (32,98,65,45,34,21,15), (32,34,45,65,98,21,15), (32,34,45,65,98,15,21), (32,21,15,98, 65,45,34) or (32,21,15,34,45,65,98). The total setup time for each of these six sequences can be calculated quickly.

We should point out that with only seven part types, an exhaustive search of all permutations would also be a feasible method for determining the part type sequence even if there were no special structure to the setup times.

7.2.2 Lot-Sizing Extensions

A dynamic program indexed to the completion of setups is developed for lot sizing in a sample problem with 10 current requirements with equal priority, where all setup times have the same probability distributions and in which setup times are the only random activities in the system (Chapter 5). By similar reasoning, a more general dynamic program can be formulated which is indexed to the occurrence of the ith random event. In addition to the secondary production requirements, this extended model is formulated to determine which, of a number of discretionary activities, the resource should be allocated to in real time. Examples of these discretionary activities include: process validation, training, unscheduled maintenance, etc.

Rather than actually formulating a new dynamic program, another model based

on an interpretation of the threshold solution is provided (Section 5.4.1). Assume that for a given level of confidence, the new model allocates a total amount of time to all required future activities (setup time, failure time, production time, scheduled downtime and time for variability in all these quantities). See Figure 7-1, but note that this is not a Gannt chart. Each time a random event occurs, we must recalculate these time allocations. More time must be allocated to each activity for a higher level of desired confidence. By subtracting the time allocated to future activities from the time left remaining in the week, we obtain an amount of available time for discretionary activities at the time random event i (*unplanned production time* or T^i_{UP}). A list of all possible discretionary activities are provided to the system in what we call the $\mathcal{P}$ *Set*. The resource may be used for any $\mathcal{P}$ *set* activity as long as it requires less than the amount of accumulated T^i_{UP}. This provides what is equivalent to the threshold. We define each of the following quantities at the time of random event i:

T^i = Time of event i.

T^i_S = Expected amount of time allocated to future setups.

T^i_F = Expected amount of time allocated to future failures.

T^i_P = Expected amount of time allocated to future production.

T^i_{ER} = Expected amount of time required for future activities.

T^i_V = Total time allocated for system variability.

T^i_{TA} = Total amount of time allocated to future activities.

T^i_{CP} = Amount of available machine time until current requirements are due.

T^i_{UP} = Unplanned production time.

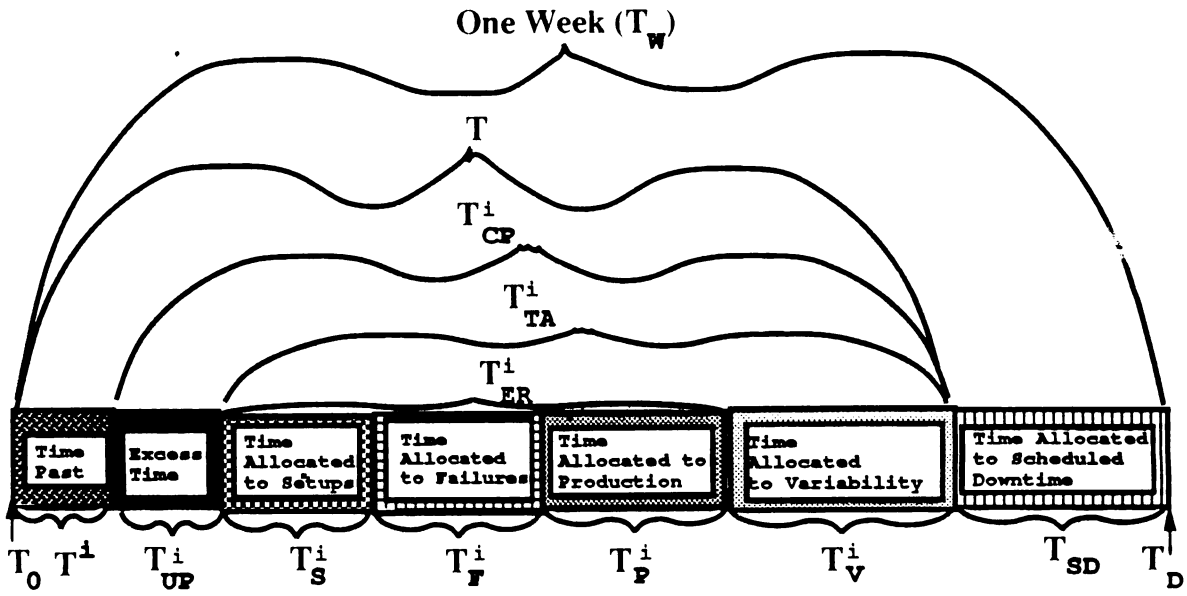

Figure 7-1: Excess Time Allocation Decision

And the following data is also provided:

T_0	=	Time orders are placed.
T_D	=	Due date (time).
T_W	=	Time from when an order is placed until it is due.
T_{SD}	=	Scheduled downtime.
T	=	Time available after scheduled downtime is accounted for.

These quantities satisfy:

$$T^i_{CP} = T - T^i \tag{7.2}$$

$$T^i_{ER} = T^i_P + T^i_S + T^i_F \tag{7.3}$$

$$T^i_{TA} = T^i_{ER} + T^i_V \tag{7.4}$$

$$T^i_{UP} = T^i_{CP} - T^i_{TA} \tag{7.5}$$

$$= T - T^i - T^i_{ER} - T^i_V \tag{7.6}$$

To determine the total time of the scheduling horizon (T_W), the time when orders are placed ($T_0 = 0$) is subtracted from the time when orders are due (T_D). Scheduled downtime (T_{SD}) is subtracted from T_W to obtain the amount of machine time available during the scheduling horizon (T). T^i is the time of the ith random event (*e.g.*, setup completion, job completion, failure completion, etc.). T^i_{ER} is the expected amount of time required for all remaining activities after the ith random event (expected production time (T^i_P) plus expected setup time (T^i_S) plus expected failure time (T^i_F)). T^i_V is the extra amount of time required to account for the variability associated with failures, yields and setup times for a given level of confidence. A normal approximation is used to estimate the value of T^i_V. However, a lower bound is set for T^i_V to account for the deterioration of the normal approximation near the end of the week. Without this lower bound, the normal approximation provides an artificially low value for T^i_V which would inflate the expectation of the system's ability to do discretionary work. T^i_{ER} and T^i_V are combined to obtain the total amount of time allocated to future activities (T^i_{TA}). Finally, *unplanned production time* (T^i_{UP} in 7.5) is obtained by subtracting total time allocated to future activities (T^i_{TA}) from total future time available (T^i_{CP}).

As events occur (*e.g.*, setups are completed, jobs are completed, failures are repaired, etc.), the actual time required to complete an activity will differ from the time previously allocated to the activity. The difference between the actual time for an activity and the time allocated to it, including time for variability, becomes available upon the activity completion. Therefore, as time passes, T^i_{UP} changes. When enough unplanned time accumulates, the machine may be used for discretionary activities. This provides a second interpretation of the threshold policy discussed in Section 5.4.1 which corresponds to the new model developed in this section. In (7.6), T^i_{UP} is directly related to T^i. For values of T^i_{UP} larger than D (time for one job), one extra job can be produced and for all values less than D, it can not. This explains the threshold result in Section 5.4.1.

The assumption that all jobs are of equal size and value is critical to this

interpretation. Having T^i_{UP} larger than D is a necessary, but not a sufficient condition, in general, for producing an extra job. For example, there may be a more highly valued job which could be completed later in the week. It might be more valuable to save the current amount T^i_{UP} so that a later job could be completed instead.

7.3 The Software

A brief description of the usage of the prototype software is now provided.

Capacity Estimation

The user initiates the session by establishing the amount of time available during the scheduling horizon (T). This is accomplished by inputting all up- and downtime for the coming week. For example, if the factory is running production 5 days per week at 10 hours per day, the user enters 50 hours of uptime into the database. The user then lists all times that the system is scheduled to be down. For example, if there is a lunch break of 1/2 hour each day (2 1/2 hours), a weekly meeting on Friday of 1 hour and scheduled maintenance on Thursday of 3 hours when the machine <u>must</u> be shut down, then there are 6 1/2 hours of scheduled downtime. This implies that there are 43 1/2 hours of uptime.

System Confidence Level

Our goal is to establish a schedule which is achievable with a certain level of confidence within a stochastic environment. As described above, the amount of time allocated to activities increases with the desired level of confidence. The

user sets the level of confidence. This is then used to determine T_V^i. Later, if the production targets cannot be met, management may choose to be less confident in meeting the schedule or add overtime. The user would go back to the previous step or the beginning of this step and revise the specified quantities.

Primary Requirements Input

Primary requirements are production requirements which must be completed within the current production period with a given level of confidence. The user inputs the current requirements one at a time until the the desired level of confidence prohibits any additional requirements. (Each time a part type is added, the system resequences the part types to minimize setup times according to the method of 7.2.1.) Assuming the user adds no overtime and does not lower the desired level of confidence (which are both system options), the general weekly profile ($\mathcal{L}$ List) is then established.

Secondary Requirements Input

Secondary Requirements are all discretionary activities for which machine time might be used (*e.g.*, the following week's production, unscheduled maintenance, process validation or even idle time). The user inputs a list of all secondary priority activities (the $\mathcal{P}$ Set). The controller automatically calculates the time required for these activities and outputs the result to the screen.

Real-Time Control

The controller is now ready for its real-time operation. It determines which activities from the $\mathcal{P}$ Set would make the maximum use of the T_{UP}^i as it becomes

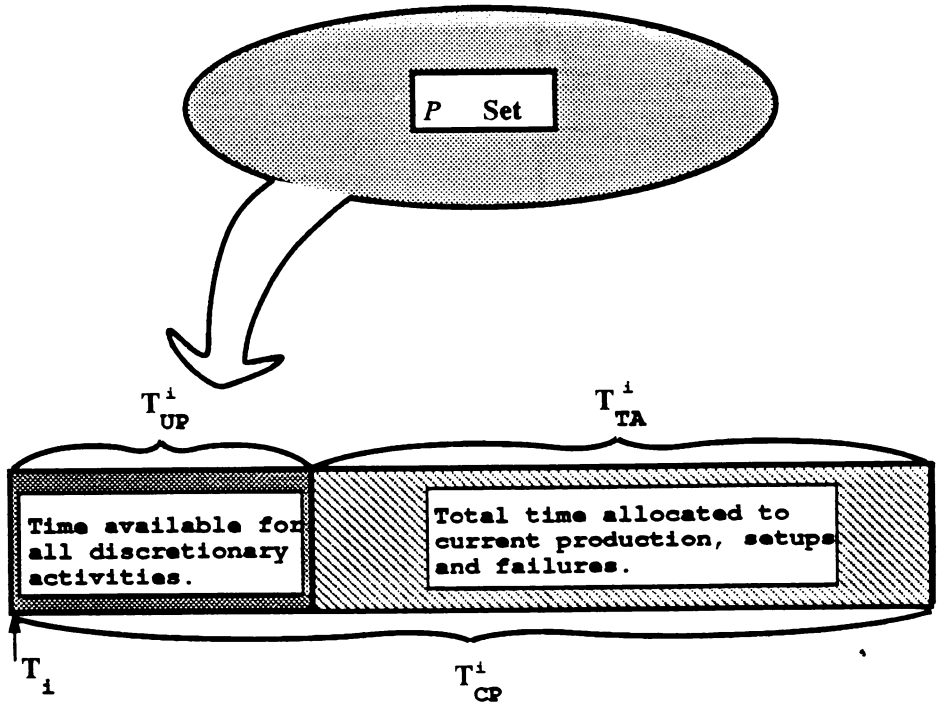

Figure 7-2: Allocation of Discretionary Activities From $\mathcal{P}$ *Set*

available. The controller constantly updates the T^i_{UP} and informs the user of the current state of the system (*e.g.*, system up, system down, T^i_{UP}, system producing P_{2312}, etc.) in real time. The user may, at any time, compare the time required for any $\mathcal{P}$ Set activity with T^i_{UP}. If the activity requires less time then T^i_{UP}, the controller advises the user to initiate it. We assume all the required data is feed to the controller by the existing CIM system.

Summary of Operation

The softwares establishes a high-level manufacturing dynamic feedback loop (Figure 7-3). Accurate system capacity and inventory information is transmitted to management. Management then sets demands which are feasible with a specified level of confidence. Then, in opposition to more traditional methods which first set

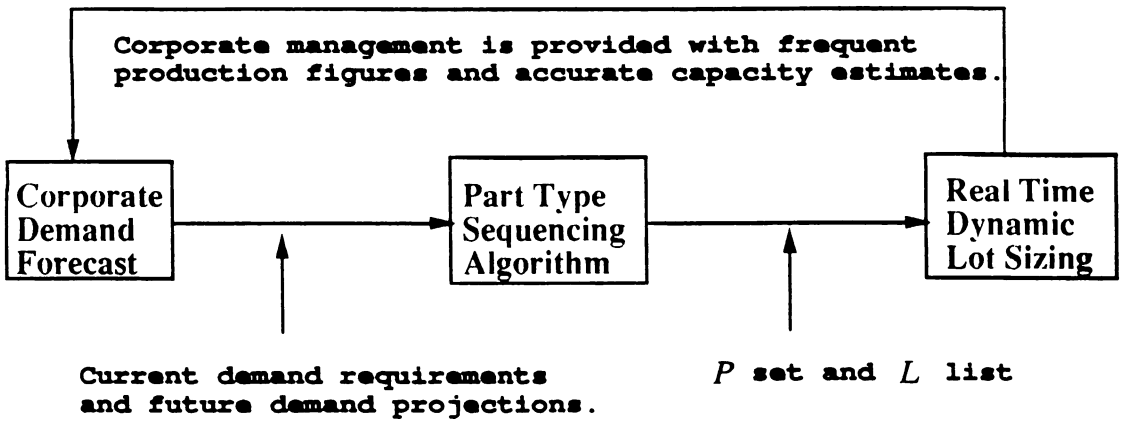

Figure 7-3: High Level Feedback Loop

lot sizes, a part type sequence is established ($\mathcal{L}$ list) along with a set of non-critical machine activities ($\mathcal{P}$ set) as inputs to an algorithm which dynamically sizes lots in response to system events. The cycle repeats as random events occur.

7.4 Future Extensions

The prototype was designed to control one machine on which a limited number of parts types are produced. It performs both the lot-sequencing and lot-sizing operations according to the principles developed in this thesis with modifications as described in this chapter. The scheduler tracks T_{UP} in real time and informs the user of potential $\mathcal{P}$ set opportunities as they become available.

Management has requested a number of extensions to the program before implementing it. They would like a more advanced method of choosing $\mathcal{P}$ set activities. They would like the software to control the starts at upstream machines. They would also like a method of employing the controller on a number of parallel ma-

chines at the bottleneck. We need to determine whether it still makes sense to decompose the problem and whether the component models will still be simple to employ. We are currently writing a proposal to do the research for these extensions.

Chapter 8

Summary and Conclusion

This thesis presents a real-time, closed-loop method of scheduling a bottleneck in a production system subject to significant, sequence-dependent, random setup times. The work is based on an empirical study of a medical device manufacturing facility (Chapter 2).

8.1 Summary

We used hierarchical decomposition to focus our research on the part dispatching function at one stage of the process (Chapter 3). A two-level hierarchy is developed which first sequences part types at the higher level and uses this sequence as input to dynamically size lots in response to system events at the lower level. This differs from more traditional approaches which first size lots and then sequence them. Scheduling the bottleneck at the dispatch level in isolation of the rest of the system is reasonable given unlimited raw materials, excess capacity downstream

and the existing distribution of event times.

At the lot-sequencing level we create what we call an $\mathcal{L}$ list which is the sequence in which part types are produced (Chapter 4). Although the facility produces many part types, the sequencing procedure is reduced to a simple greedy algorithm. Under ideal conditions this algorithm provides the minimum total setup time sequence. A bound on the difference between the true minimum time and that obtained by using the greedy algorithm is also attainable. The advantages of this method include its high speed, its ability to be programmed easily, its low data requirements and the ease with which factory personnel could understand it.

Given the $\mathcal{L}$ list, a lot-sizing algorithm is then employed at Level 2 (Chapter 5). The problem is formulated as a dynamic program. A sample problem is presented to test the model and generate a control policy. The resulting partitioned sample space suggests a threshold policy. This threshold is also very similar to a conservative boundary threshold which is easy to calculate. Therefore, a heuristic threshold policy based on the conservative boundary is suggested. Simulation results show that the heuristic performs very well compared with current industrial practices.

A prototype software package was designed to control an experimental testbed machine at the factory. A variable called T_{UP}^{i} or *unplanned production time* is created representing the amount of residual time available at any time T_i for doing non-current requirements on the bottleneck machine. The i superscripts indicates that this is the amount of unplanned production time accumulated at the time of the ith random event. With the passing of each event, T_{UP}^{i} changes. In addition, a $\mathcal{P}$ set of alternative activities is presented. T_{UP}^{i} is continually compared to the times required for activities in the $\mathcal{P}$ set. If only activities which require less than T_{UP}^{i} are initiated at any point in time, the system operator can remain confident that the system will complete the current requirements with a high level of confidence. This allows a real-time decision of what to do with the bottleneck machine.

8.2 Future Work

As discussed in the end of Chapter 7, a full implementation requires a number of theoretical extensions including:

1. Testing different cost functions to account for different business situations.
2. Including failures and other random activities in the analysis in a more rigorous way than was presented in Chapter 7.
3. Extending the model to multiple serial and parallel machine cases.
4. Accounting explicitly for up- and downstream control.
5. Providing more general rules for optimizing the selection of $\mathcal{P}$ set activities.

8.3 Conclusion

The results of our work demonstrate that closed-loop feedback is a reasonable methodology for factory scheduling subject to uncertainty. While our theoretical models account only for setup randomness, our implementation suggests that similar concepts can be employed for accounting for many other random phenomena. The experience of conducting this research has demonstrated the need for both intellectual and practical extensions to the current manufacturing scheduling paradigm.

Bibliography

[1] H. Bahl, L. Ritzman, and J. Gupta. Determining lot sizes and resource requirements: A review. *Operations Research*, volume 35(No. 3):pages 329–345, (May-June) 1987.

[2] K. Baker and G. Scudder. Sequencing with earliness and tardiness penalties: A review. *Operations Research*, volume 38(No. 1):pages 22–36, (January-February) 1990.

[3] D. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice Hall, New Jersey, USA, 1987.

[4] G. Bitran and A. Hax. On the design of hierarchical production planning systems. *Decision Sciences*, volume 8:pages 28–55, 1977.

[5] E. Burch, M. Oliff, and R. Sumichrast. Linking level requirements in production planning and scheduling. *Production and Inventory Management Journal*, pages 123–131, Second Quarter 1987.

[6] M. Burman. Recent trends in production research. Technical report, In Preparation, MIT, 1992.

[7] Y. Chang, H. Matsuo, and R. Sullivan. A bottleneck-based beam search for job scheduling in a flexible manufacturing system. *International Journal of Production Research*, volume 27(No. 11):pages 1949–1961, 1989.

[8] O. Charles-Owaba and B. Lambert. Sequence dependent machine set-up times and similarity of parts: A mathematical model. *IIE Transactions*, volume 20(No. 1):pages 12–21, March 1988.

[9] S. Connolly. A real-time policy for performing setup changes in a manufacturing system. Master's thesis, MIT, 1992.

[10] H. D'Angelo, M. Caramanis, S. Finger, A. Mavretic, Y. Phillis, and E. Ramsden. Event-driven model of unreliable production lines with storage. *International Journal of Production Research*, volume 26(No. 7):pages 1173–1182, 1988.

[11] D. Dilts and K. Ramsing. Joint lot sizing and scheduling of multiple items with sequence-dependent setup costs. *Decision Sciences Journal*, volume 20(No. 1):pages 120–133, Winter 1989.

[12] S. Elmaghraby. The economic-lot scheduling problem (ELSP): Review and extensions. *Management Science*, volume 24(No. 6):pages 587–597, 1978.

[13] B. Fleischmann. The discrete lot-sizing and scheduling problem. *European Journal of Operations Research*, volume 44:pages 337–348, 1990.

[14] T. Galvin. Economic lot scheduling problem with sequence-dependent setup costs. *Production and Inventory Management Journal*, pages 96–105, First Quarter 1987.

[15] S. Gershwin. A hierarchical framework for manufacturing system scheduling: A two-machine example. In *The 26th IEEE Conference on Decision and Control, LA, CA*, 1987.

[16] S. Gershwin. Hierarchical flow control: A framework for scheduling and planning discrete events in manufacturing systems. *IEEE, Special Issue on Dynamics of Discrete Event Systems*, volume 77(No. 1), 1989.

[17] S. Gershwin. *Manufacturing Systems Engineering.* Prentice Hall, Inc., 1993. In Preparation.

[18] S. Gershwin, M. Caramanis, and P. Murray. Simulation experience with hierarchical scheduling for a simple manufacturing system. In *27th IEEE Conference on Decision and Control*, 1988.

[19] E. Goldratt and J. Cox. *The Goal, A Process of Ongoing Improvement.* North River Press, Inc., New York, 1986.

[20] D. Golhar and C. Stamm. The just-in-time philosophy: A literature review. *International Journal of Production Research*, volume 29(No. 4):pages 657–676, 1991.

[21] J. Gooch, M. George, and D. Montgomery. *America Can Compete.* The Institute of Business Technology, Dallas, TX, 1987.

[22] S. Graves. A review of production scheduling. *Operations Research*, volume 29(No. 4):pages 646–675, (July-August) 1981.

[23] R. Hayes, S. Wheelwright, and K. Clark. *Dynamic Manufacturing: Creating the Learning Organization.* The Free Press, NY, USA, 1988.

[24] L Johnson and D. Montgomery. *Operations Research in Production Planning, Scheduling, and Inventory Control.* Wiley, New York, 1974.

[25] A. Jones and C. McLean. A proposed hierarchical control model for automated manufacturing systems. *Journal of Manufacturing Systems*, volume 5(No. 1):pages 15–25, 1986.

[26] R. Karni. A study of lot-sizing algorithms. *IIE Transactions*, pages 356–366, December 1986.

[27] L. Kleinrock. *Queueing Systems.* John Wiley and Sons, NY, 1975.

[28] M. Liberatore and T. Miller. A hierarchical production planning system. *Interfaces*, volume15(No.4):pages 1–11, (July-August) 1985.

[29] S. Melnyk and P. Carter. Scheduling, sequencing and dispatching: Alternative perspectives. *Production and Inventory Management*, pages 58–67, Second Quarter 1986.

[30] C. Monma. On the complexity of scheduling with batch setup times. *Operations Research*, volume 37(No. 5):pages 798–804, (September-October) 1989.

[31] P. Neeley. Scheduling on the edge of turbulence. *Production and Inventory Management Journal*, pages 10–14, Second Quarter 1990.

[32] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Inc., NJ, 1982.

[33] N. Pence, J. Megeath, and J. Morrell. Coping with temporary bottlenecks in a several-stage process with multiple products. *Production and Inventory Management Journal*, pages 5–7, Third Quarter 1990.

[34] F. Rodammer and K. White, Jr. A recent survey of production scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 18(6):pages 841–851, (November-December) 1988.

[35] A. Sharifnia, M. Caramanis, and S. Gershwin. Dynamic set-up scheduling and flow control in manufacturing systems. *Journal of Discrete Event Dynamic Systems*, volume 1(No. 2), 1991.

[36] T. Shioyama and H. Kise. Optimization in production systems: A survey on queueing approaches. *Journal of the Operations Research Society of Japan*, volume 32(No. 1):pages 34–55, 1989.

[37] V. Sridharan and W. Berry. Freezing the master schedule under demand uncertainty. *Decision Sciences Journal*, volume 21(No. 1):pages 97–120, 1990.

[38] N. Srivatsan and S. Gershwin. Selection of setup times in hierarchical controlled manufacturing system. In *Twenty-Ninth IEEE Conference on Decision and Control*, 1990.

[39] C. Tang. Scheduling batches on parallel machines with major and minor setups. *European Journal of Operational Research*, volume 46:pages 28–37, 1990.

[40] F. Taylor. *Scientific management (Comprising Shop Management, The Principles of Scientific Management, and Testimony Before the Special House Committee).* Harper & Brothers Publishers, New York, NY, 1947.

[41] S. Taylor and S. Bolander. Scheduling product families. *Production and Inventory Management*, pages 47–55, Third Quarter 1986.

[42] W. Trigeiro. A simple heuristic for lot sizing with setup times. *Decision Sciences Journal*, volume 20(No. 2):pages 294–303, Spring 1989.

[43] W. Trigeiro, L. Thomas, and J. McClain. Capacitated lot-sizing with setup times. *Management Science*, volume 35(No. 3):pages 353–366, March 1989.

[44] J. Violette and S. B. Gershwin. Decomposition of control in a hierarchical framework for manufacturing systems. In *IEEE Conference on Automatic Control, Boston*, 1991.

[45] H. Wagner and T. Whitin. Dynamic version of the economic lot size model. *Management Science*, volume 5(No. 1):pages 89–96, October 1958.

[46] K. Zoller and A. Robrade. Efficient heuristics for dynamic lot sizing. *International Journal of Production Research*, volume 26(No. 2):pages 249–265, 1988.