SIMEX:
A Simulation-based Expert Production Scheduling System

By
Risvan Coskun

A thesis
submitted to the School of Graduate
Studies and Research of the University of Ottawa
in partial fulfillment of the requirements of
Master of Science in Systems Science

University of Ottawa
Ottawa, Ontario
Canada
Nov 1994

ISBN 0-612-04868-3

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this document. I authorize the University of Ottawa to lend this document to other institutions or individuals for the purpose of scholarly research.

Risvan Coskun

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

Risvan Coskun

To my parents: En içten sevgilerle.

TABLE OF CONTENTS

Acknowledgements

I would like to express my gratitude to my thesis supervisor, Dr. Tuncer I. Ören, for his valuable time, guidance and advise during the research and preparation of my thesis.

I also would like to express my thankfulness to my parents Atila and Gönül Coskun, my sister Aslihan Battal, and my brother Yildirim Beyazit Coskun for their endless support.

Trademark and Copyright Acknowledgement

ABSTRACT

A good methodology for production scheduling can result in high efficiency in reducing manufacturing costs. Conventional Operations Research approaches are often insufficient for scheduling problems due to the combinatorial complexity of the search space, uncertainty in manufacturing environments, and conflicting objectives in the optimization criteria (Jürgen et al., 1992). In order to fully support a manufacturing system to carry out a production scheduling, an interactive, user friendly and fast production scheduler is desirable. Moreover, a real time scheduling system has to solve complex scheduling problems, which do not respond to conventional algorithmic solutions, with the help of heuristic approaches, with numerous options and factors (Arons, 1990). In recent years, simulation and knowledge-based system technologies have been successfully applied to production scheduling problems to satisfy this demand. SIMEX is an experimental simulation-based expert production scheduler developed by the author for applications in flexible flow shop systems in a dynamic factory environment. This study introduces the general framework of SIMEX. A prototype is developed on an IBM compatible PC in Prolog, MODSIM II, Visual Basic, and Visual C++ to generate feasible and acceptable schedules with a synchronous data exchange facility.

In general, primary tasks of SIMEX are to meet due dates of the final products, to increase throughput by reducing the number of setups, and to reduce inventory cost in a flexible flow shop system in real time. SIMEX has also an ability to change its expert system's rule base interactively by means of a user interface. The expert system module of SIMEX allows to use heuristics, and production rules which are the simplifications that help limit the search for possible problem

solutions and handling unexpected events. Simulation-based scheduler written in MODSIM II, is another module of SIMEX. It generates the schedules, repeatedly, to analyze and verify proposed design and alternatives.

ABBREVIATIONS

CABINS	Case-based Interactive Scheduler
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CBR	Case-based Reasoning
CDS	Campbell, Dudeh and Smith Algorithm
CIM	Computer Integrated Manufacturing
EDD	Earliest Due Date
FOL	First-Order Logic
IME	Intelligent Manufacturing Engineering
JIT	Just-in-Time
LITL	Longest in-process Idle Time Last
LTTL	Longest Terminal Time Last
MHS	Material Handling Systems
MICRO-BOSS	Micro-Bottleneck Scheduling System
MLDF	Maximum Level of Depth First
MPS	Master Production Schedule
MRP	Material Requirement Planning
MRP II	Manufacturing Resource Planning
NC	Numerical Control
NNH	Nearest Neighbor Heuristic
NPS	*N* Product Scheduling
OOP	Object Oriented Programming
PICS	Production and Inventory Control System
PMS	Production Management Systems
PSS	Production Scheduling System
SFC	Shop Floor Control
SPC	Statistical Process Control
SPS	Single Product Scheduling
SPT	Shortest Processing Time
TQC	Total Quality Control
TSP	Traveling Salesperson Problem
WIP	Work-in-Process

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Basics

The solution of a production scheduling problem is any feasible resolution of two types of constraints: (1) limited resources to be allocated to perform each task, and (2) a distinct time period for each task to be performed. Therefore, "solving" a production scheduling problem amounts to figuring out which resources will be allocated to perform each task and when each task will be performed in an optimal manner with respect to some criterion.

There are several types of production scheduling. They are:

- Conventional production scheduling
- Knowledge-based production scheduling
- Simulation-based production scheduling
- Simulation-based expert production scheduling

Conventional production scheduling models or algorithms aim to generate an optimal solution by following a set of rules and steps which exactly determine the final schedule. Although this approach might be thought as an extension of operations research applications in order to find optimum schedules for production scheduling problems, unfortunately due to complexity of real life problems, only a few simple cases this method leads useful results. Complexity of real life problems may be named literally such as: (1) complexity of manufacturing environment constraints, (2) complexity of relationships between objects, and (3) unexpected events in a dynamic manufacturing environment (i.e., machine break-downs, material shortage, etc.).

In order to eliminate the complexity of real life problems, it is suggested to use priority rules when generating schedules (Blackstone et al. 1983). One dispatch rule, for example, recommends to schedule the operation with Least Processing Time (LPT) first, while another rule uses Earliest Due Date (EDD) to prioritize operations. Although it is a practical method of obtaining schedules and computationally cheap, the solution could and will often be far from optimal, because there is no dispatching rule which is effective for all scheduling problems. The results obtained for a given rule differ according to the operating conditions. It is therefore important to know the type of assumptions and operating conditions used before dispatching rules can be selected for proper applications.

Arons (1990) emphasizes that conventional production scheduling methods and priority rules lack human experience. This lack is generally considered as a drawback. He says that even anybody with a little experience in manufacturing do have the ability to make good dispatching decisions. In fact, interactive scheduling originates from this idea. Decision maker may make dispatching decisions interactively whenever necessary to keep production on with his experience and familiarity to the system.

Knowledge-based production scheduling is one of the latest trend which includes human expertise to the system itself when generating production schedules, instead of interactively interfering to the dispatching decisions. In literature, there are a lot of recent papers issued on the subject. Fox and Smith (1984), Erschler and Esquirol (1986), Smith (1988), Kusiak (1991), Miyashita and Sycara (1992) and Sadeh (1992) suggested to use knowledge-based systems, and they developed their own knowledge-based production scheduling systems. They all agree that symbolic reasoning has dominant advantageous compare to traditional programming where applications of heuristic solution techniques do not respond to algorithmic solution techniques. By a knowledge-based system, significant amount of qualitative features of a situation can be captured and manipulated

rather than relying on numeric solutions. And by meta-level languages, more sophisticated control of a problem can be achieved.

Simulation-based production scheduling is suggested by MacFarlan and Grant (1990), Wyman (1991), and Mebarki (1993). They state that conventional production scheduling models are not sufficient to represent a realistic factory model. Because conventional models have a lot of simplifications on manufacturing environment constraints to make them mathematically tractable. On the contrary, simulation models have no such restrictions

Simulation-based expert production scheduling is due to Bruno et al. (1986), Swaan Arons 1990, and Fargher and Smith (1992). They suggested to use simulation techniques and knowledge-based system technologies together. They showed that such a combination would result in realistic factory modeling, unexpected event handling, human expertise on decision making, and more important, obtaining improved production schedules in real time. In this paper I propose an approach based on real-time response which attempts to address each of the above problems by using simulation and knowledge-based systems technologies.

1.2 Presentation of the Content of the Thesis

The aim of the research presented in this thesis is to extend the knowledge on scheduling problems that require to handle unexpected events in a dynamic manufacturing environment. More precisely, we are interested in studying how the knowledge of unexpected events with the resource constraints can be used to produce detailed production schedules which provide better time and resource management, in terms of improved production control.

Under this objective, a general framework of a production scheduling system and a heuristic algorithm to be used in the system are developed. This system is a real time industrial production scheduler with some desirable properties such as (1) user friendly interface and reporting, (2) interactive scheduling, (3) interactive shop floor control, (4) inclusion of human expertise, and most importantly (5) unexpected event handling.

The remainder of this thesis is divided in eight chapters. In Chapter 2, historical development of manufacturing systems from 1960s to 1990s and an overview of production scheduling problem are introduced. Basic notation and important definitions from the literature are given.

In Chapter 3, the most popular conventional production scheduling systems (PSS) are introduced. Scheduling n operations on a single machine, two-machine flow shop problem, two-machine job shop problem, special cases of a three-machine flow shop problem, M-machine flow shop model, single product scheduling problem, and P-product scheduling problem are discussed.

An overview of knowledge-based production scheduling systems are given in Chapter 4. In addition to that, basic concepts of knowledge-based systems, common features of knowledge-based production scheduling systems, and manufacturing knowledge representation are discussed with examples from the literature.

An overview of simulation-based production scheduling systems are given in Chapter 5. In addition to that, basic concepts of simulation, and common features of simulation based production scheduling systems are discussed from the literature.

In Chapter 6, an overview of simulation-based expert production scheduling systems are given.

In Chapter 7, SIMEX which is a simulation-based expert production scheduling system is introduced. The real life problem definition are given. The solution approach, features of SIMEX, and system components of SIMEX are explained. The driving force behind this chapter comes from the previous chapters 4, 5, and 6.

In Chapter 8, implementation results are displayed.

Chapter 9 is the conclusion of the thesis. The major contributions are summarized, and the shortcomings of our system and future work are discussed.

2 MANUFACTURING SYSTEMS AND PRODUCTION SCHEDULING PROBLEM

Today, modern industry deals with more complicated products than it did in the last several decades. As sophistication of the product increases, the number of operations, and thus complexity of production scheduling increases as well. Although the manufacturing tools are advanced parallel to the other developments on the technology, sophisticated products, complexity of production scheduling, and uncertainty in the factory environment caused engineers to fail to achieve valid production schedules obtained with the help of conventional techniques.

The next section introduces a short historical background of modern manufacturing systems and the fundamental problems in manufacturing. After that, scheduling problem will be discussed in the context of production.

2.1 Historical Development of Manufacturing Systems: After 1960s

In the 1960s, the primary competitive thrust was cost. Thus, the accepted manufacturing strategy was based upon cost minimization in high volume production with stable plant conditions. Production and inventory control system (PICS) was the major tool to manage manufacturing environments with numerical control (NC) machine tools.

In the 1970s, the primary competitive thrust had shifted to marketing, although cost was an important dimension. The manufacturing strategy turned out to be more directed toward functional integration. During this period, manufacturing systems became more integrated across traditional organizational boundaries. Material requirement planning (MRP) grew out of the original PICS concepts. MRP was thought to be a unique

planning system which provided execution of shop floor control (SFC). Master production schedule (MPS) was seen as a critical factor for system design and operation to control of shop floor (e.g., a driving force behind MRP).

In the 1980s, the primary competitive thrust had shifted to quality. Manufacturing strategy changed into process control, world class manufacturing, overhead cost reduction and material velocity. The manufacturing system approach was to improve quality with statistical process control (SPC), and total quality control (TQC) techniques. Computer aided design (CAD) and computer aided manufacturing (CAM) started to play roles in the manufacturing arena because of advanced developments in computer technology.

In this period, MRP evolved into MRP II, which is manufacturing resource planning. The goal of the evolution was to include more company activities and functions in the system. Cost accounting, financial planning and budgeting were integrated to the system and production planning became a top management activity.

The concept of "material velocity" grew out of the just-in-time (JIT) approach which has a somewhat different and new influence on manufacturing systems. One of the fundamental objectives of JIT is to increase material velocity. The goal is to move products through manufacturing so quickly that one does not need to consider the status of work-in-process (WIP) (Ebner and Vollman, 1988).

2.2 Manufacturing Challenges of the 1990s

The 1990s will be different in several important dimensions. Direct labor as a source of value to the products and services will continue to decrease in relative importance. Knowledge workers will be the creators of what is provided to customers. These

knowledge workers are the creators of the features of the company that differentiate it in competitive environment. For the long run survival and productivity, the effectiveness of these knowledge workers is very critical (Ebner and Vollman, 1988).

In order to increase the effectiveness of the knowledge workers, successful integration of knowledge is very important. Computer integrated manufacturing (CIM) with this intention evolved from MRP II concepts. CIM is strategically important to support new product development and production control faster (Ebner and Vollman, 1988).

In the competition arena, time will be the major constraint. This is the primary competitive motive of business in the 1990s. It does not mean that other motives can be passed over. Cost and quality are also obligatory to remain in the race. But the winning factor is endowed by time and enhancements of the basic products. At the factory level, time-based competition becomes the highest priority to gain responsiveness and flexibility.

The driving force behind this priority setting is the need and the wish to respond to virtually any customer request just in time. On the other hand, flexibility, which is the response of a system to environmental uncertainties (e.g. unknown customer, machine breakdowns, quality failure, etc.), will be one of the hardest possession that a factory must maintain a certain level without contradicting the productivity level (section 2.3). Unexpected events may invalidate default production schedules, and therefore adjustments might be required. The required response may be achieved by knowing how to manage uncertain knowledge (Meyer, 1990).

2.3 Fundamental Problems in 1990s' Manufacturing

In this section, I will summarize the basic problems of modern manufacturing environments. As complexity of products increases, uncertainty of the manufacturing environment also increases due to the technological requirements to perform operations of the products. Therefore, manufacturing firms are trying to balance themselves on the opportunity line between productivity and flexibility.

Productivity versus flexibility

Complexity of knowledge and uncertainty of manufacturing environment conditions are the main factors which cause some delay due to investigation and solving of the problems occurred. The fundamental trade-off curve of productivity versus flexibility, or cost versus time is heavily influenced by these factors and only qualitatively known for a specific plant. One must constantly follow these curves and define the production control mode (Figure 2.1) (Meyer, 1990).

Complexity

Any task can be broken down into a number of subtasks and operations. Such task disaggregation might increase the productivity. When different subtasks are performed by different workers the labor itself has become divided. But further division of labor is restricted by coordination costs which exceed the productivity gain.

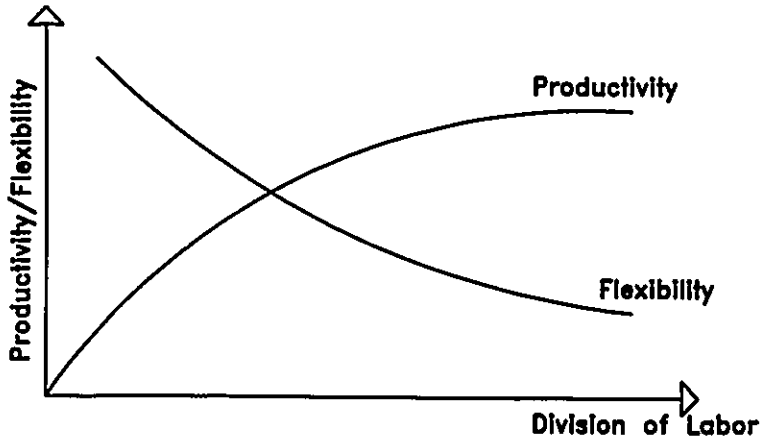


Figure 2.1 The basic dilemma in manufacturing: gain in productivity versus loss in flexibility

This holds equally for physical labor and expert knowledge which are the two foundations of the economic resources of a company. The specialization of knowledge is similar to the disaggregation of the labor.

Today, the JIT efforts which aim at flow-oriented site layout, concern the reintegration of physical labor (equipment/machines) via flow lines, whereas CIM anticipates the reintegration of special knowledge organized in functional departments by integrated information processing. Here the process knowledge from the factory floor is made available to the product designer, to reduce product complexity. In fact, the amount of shared knowledge determines the degree of cooperation between team tasks. The cooperation costs, therefore, are communication costs (Figure 2.2) (Meyer, 1990).

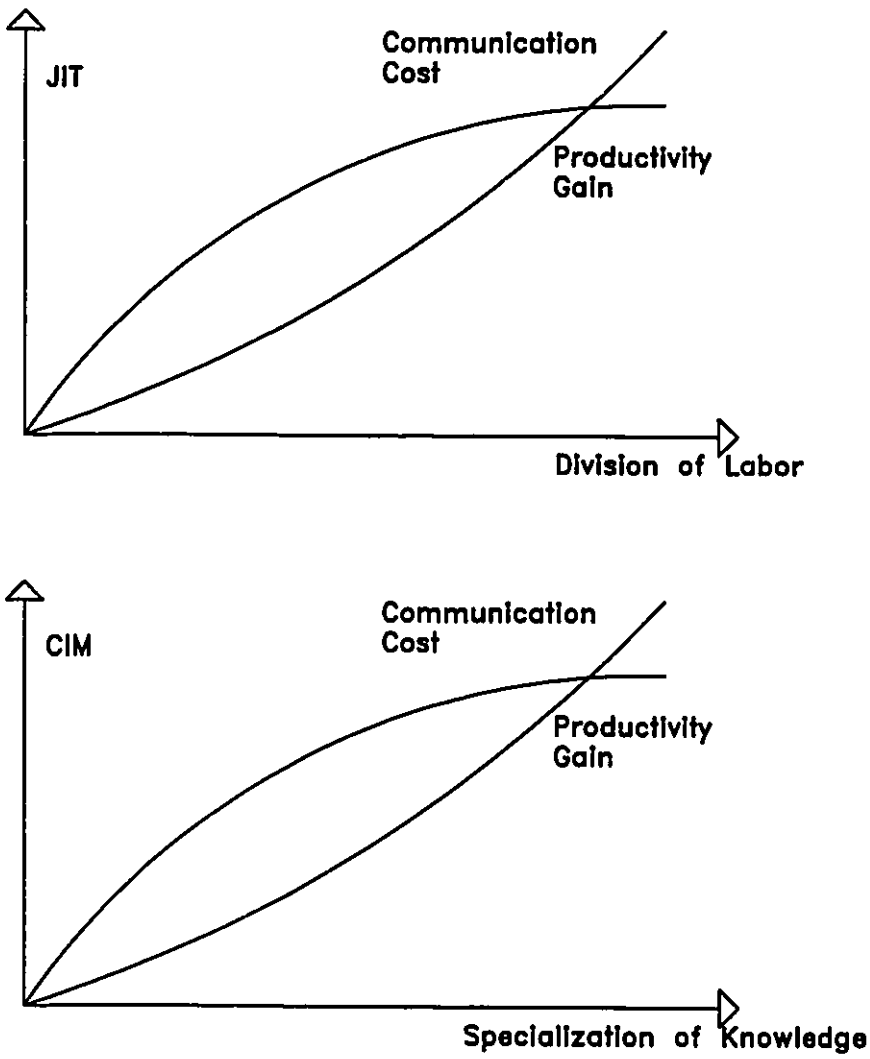


Figure 2.2 The two basic approaches to the diversification problem may be given as
- integration of physical labor (JIT)
- integration of knowledge contained in functional departments (CIM)

Uncertainty

A dynamic and uncertain factory environment needs planning and control. There are three stages for dealing with the uncertainty. First one is adaptation which specifies some environmental parameters for a period of time. Second one is optimization of control parameters. Third one is the control in real life. For example; production scheduling

should be optimized for increasing the system responsiveness to demand in order to (1) meet due dates, (2) reduce total flow time, (3) balance factory loads, etc. (Meyer, 1990).

This is the planning part of the problem. At the second stage, the control problem occurs when the planning is executed, such as machine scheduling, lot splitting and job sequencing, that is with:

- exploiting resources effectively
- respecting due dates in the face of uncertainty.

The model for the control of a problem can be given as the following Figure 2.3:

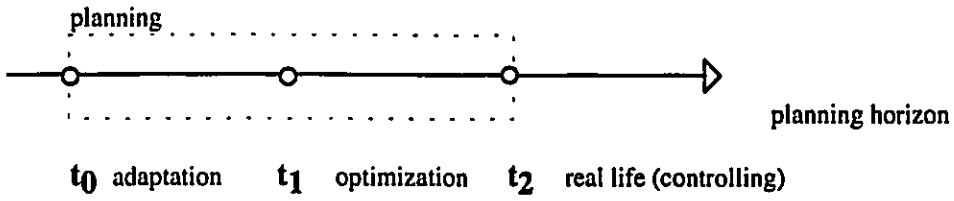


Figure 2.3 Model for the control of a problem

Origin of the manufacturing problems:

The origin of the manufacturing problems lies in the complexity of the task disaggregation and uncertainty of the dynamic environment in which decisions are made (Figure 2.4). A decision is a transformation process of information into action. Indeed the behavior of a decision maker is determined primarily by the knowledge and messages received from the environment, s/he operates on models of the environment, not on real things. Instead of work, they manipulate the symbols of working (e.g. speech, memos, meetings, figures, etc.) (Meyer, 1990).

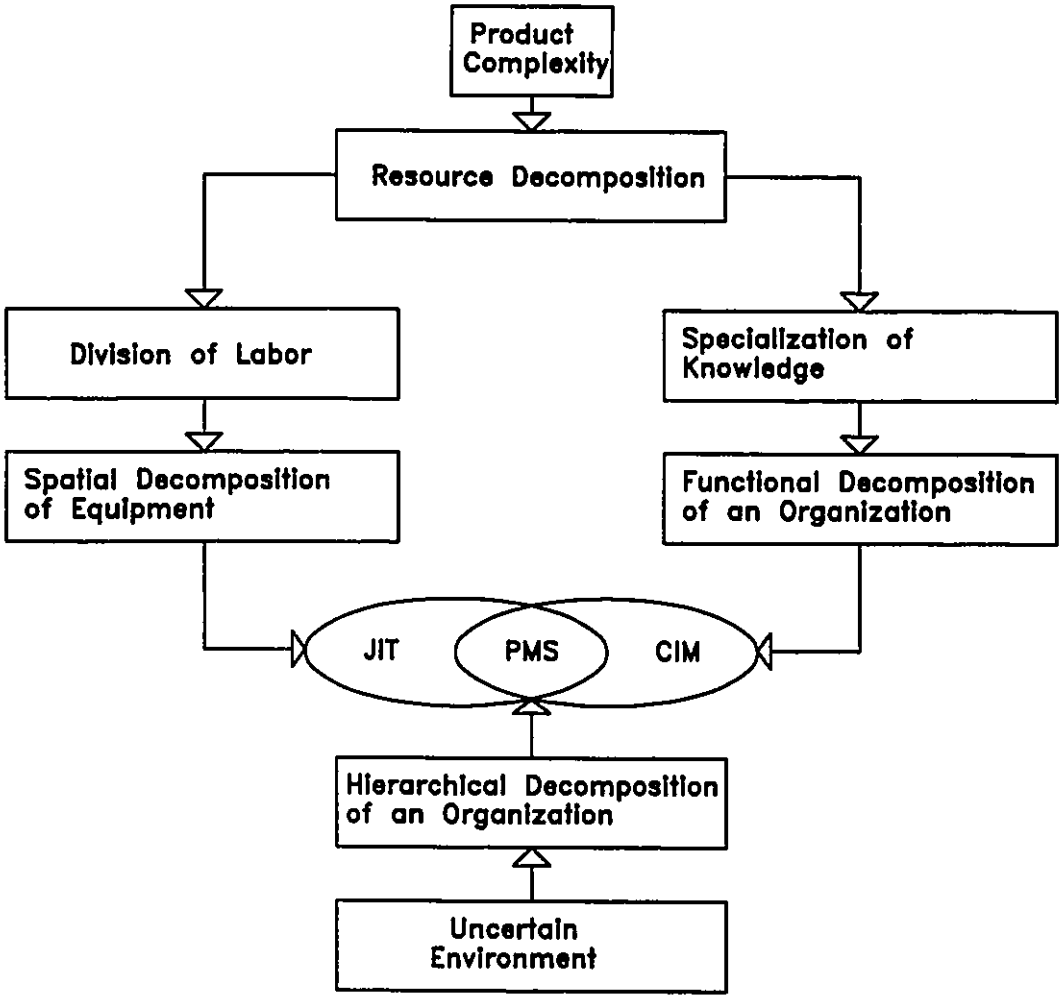


Figure 2.4 JIT : reintegrate labor; CIM : reintegrate knowledge; PMS : cope with uncertainties from market and plant (Meyer, 1990)

2.4 An Overview of Production Scheduling Problems

In general, there are two kinds of production scheduling problems: (1) Operation scheduling, and (2) Product scheduling. In this section, I will introduce the concept of both types of the production scheduling problems.

2.4.1 Operation Scheduling

A brief definition of the deterministic production scheduling problem is paraphrased from (Kusiak, 1990) as follows:

A set of products is to be processed on a set of machines in order to minimize or maximize a certain performance measure. A product necessitates a number of operations. All machining parameters are assumed to be known in advance. Each operation is to be processed by at most one machine at a time.

In this kind of scheduling models, resources are usually called machines and tasks are called jobs. A job may consist of a number of operations. All machining parameters are assumed to be known in advance. Each operation is to be processed by at most one machine at a time. There are two modes of processing depending on the way the products visit machines: flow shop and job shop (see Figure 2.5).

In the flow shop, all parts flow in one direction, whereas in the job shop, the parts may flow in different directions. In particular, each operation after the first has exactly one direct predecessor and each operation before the last has exactly one direct successor. Therefore, each job requires a specific sequence of operations to be carried out for the job to be complete. The shop contains m different machines, and n different operations, each of which requires a different machine. The machines can thus be numbered 1, 2, ..., m; and the operations of job j can be numbered $(1, j)$, $(2, j)$, ... , (m, j), $n = m$, with the corresponding machine number.

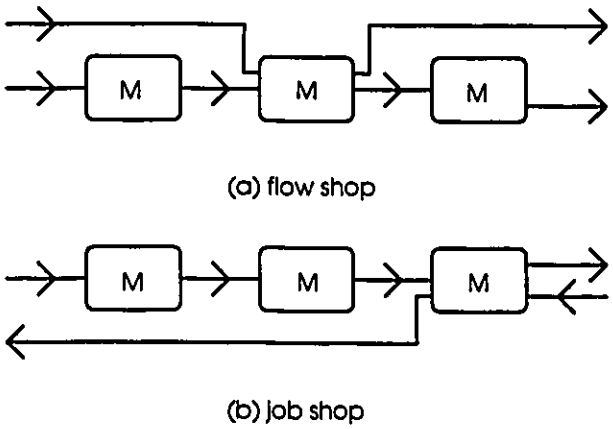


Figure 2.5 Illustration of two models of processing: "M" denotes machines, "→" denotes the flow of products

In the job shop, flow of work is not unidirectional. Unlike the flow shop model, there is no initial machine that performs only the first operation of a job, nor is there a terminal machine that performs only the last operation of a job. In the flow shop, machine k performs operation k of any job, and there is no need to distinguish between operation number and machine number. By contrast, in the job shop, description of an operation is given with a triplet (i, j, k) which denotes operation i of job j requiring machine k.

In a production scheduling problem, the objective function consists of all costs in the system that depend on scheduling decisions. In practice, such costs are difficult to identify completely, in fact, the major costs can be determined from the most observable ones. In principal, there are three types of goals: *turnaround*, *timeliness*, and *throughput*. Turnaround measures the time required to complete a product. Timeliness measures the ability of a particular product's completion to a given deadline. Throughput measures amount of product completed during a fixed period of time.

A production scheduling problem model may contain one machine or several machines. If it contains only one machine, then one can conclude that most of the products may be

completed with a single operation, while multiple machines usually involve products with multiple operations. In both cases, machines may be available in unit amounts or in parallel.

Single machine problem may be categorized as with/without due dates. If due dates are not imposed, one can minimize the total flowtime of the product which is the addition of the times for all job *j* spent in the system. Here the scheduling objective is rapid turnaround which reduces work-in-process inventories related the time spent by a job in the system and early completion of the job is desirable. By trying to minimize total flow time, and hence, we can simultaneously minimize average work-in-process inventory level. If due dates are imposed, in such circumstances, one can minimize maximum lateness or maximum tardiness. Emmons (1969), Glover (1989), Kusiak (1990), Lawler (1977, 1982), Moore (1968), Potts and Wassenhove (1982), and Shwimer (1972), described optimization and heuristic methods to schedule by minimizing maximum tardiness or lateness on a single machine.

A static production scheduling problem model has a set of jobs which does not change over time. In contrast, a dynamic production scheduling problem model has a set of jobs which may change over time. New jobs may appear, or some jobs may disappear. As a result of the dynamic set, number of operations belonging to a job may change in time.

There are mainly two kinds of feasibility constraints in production scheduling problems: capacity of machines and order of the operations to be performed.

If the production scheduling problem contains two or more machines, the system may be either a flow shop model, or a job shop model. For two-machine flow shop model, Johnson (1954) described how to minimize the makespan (or schedule length). He

proved that for an optimal sequence of operations, an operation with the shortest processing time and corresponding machine must be selected first. Campbell et al. (1970) described a heuristic, called CDS algorithm, which uses Johnson's rule in a heuristic fashion and creates several schedules from which a best schedule can be chosen.

For two-machine job shop model, Kusiak (1990) described an algorithm to minimize the maximum flowtime. He uses Johnson's algorithm in a heuristic fashion and obtains an optimal schedule.

Three machine flow shop model, for several special cases, can be solved efficiently, yet it is difficult to generalize a result like two-machine flow shop model. Burns and Rooker (1976, 1978) and Szwarc (1977) described those special cases to find optimum without enumerative search.

The solution model for n operations on m machines is due to Baker (1974). The model is mainly a mixed-integer programming problem.

2.4.2 Product Scheduling

Operation scheduling can be thought as a pure machine scheduling problem. On the contrary, product scheduling differs from operation scheduling by the idea of aggregate production scheduling. Kusiak (1991) introduces an example of aggregate production scheduling. He proposes a manufacturing system which consists of a machining subsystem and an assembly subsystem. The two subsystems are linked by a material handling carrier. Consider an example product P with parts to be machined and then assembled. It consists of subassembly A_1, and final assembly A_2, and three parts P_1, P_2, P_3. The precedences among machining and assembly operations for the product can be represented by a directed graph (digraph) shown in Figure 2.6.

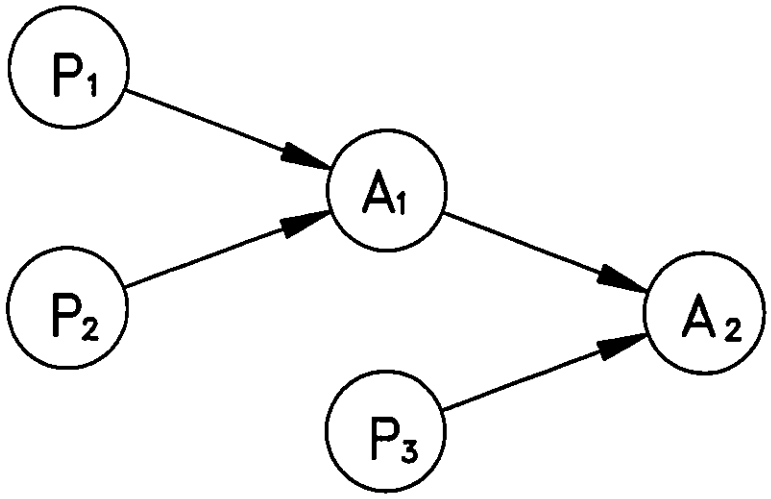


Figure 2.6 A digraph of the example product P

In this digraph, any node of degree 1, i.e., with the number of edges incident to the node equal to 1, denotes a part; any node of degree greater than 1 denotes a subassembly or a final product.

Regarding to the digraph, parts P_1 and P_2 are to be machined before the subassembly A_1 is obtained. Assembling P_3 and A_1 results in the product P (or final assembly A_2). A possible final schedule is shown in Figure 2.7.

2.5 Computational Complexity

General theory of algorithms includes three large subject areas regarding computational properties of problems: (1) the theory of computability, (2) design of algorithms, and (3) the theory of computational complexity (Harel 1987). An algorithm is a set of instructions, expressed in some language, for executing a sequence of operations for solving a problem of some specific type. Algorithms are required to be finite, i.e., each algorithm must terminate after a finite number of steps or operations have been executed.

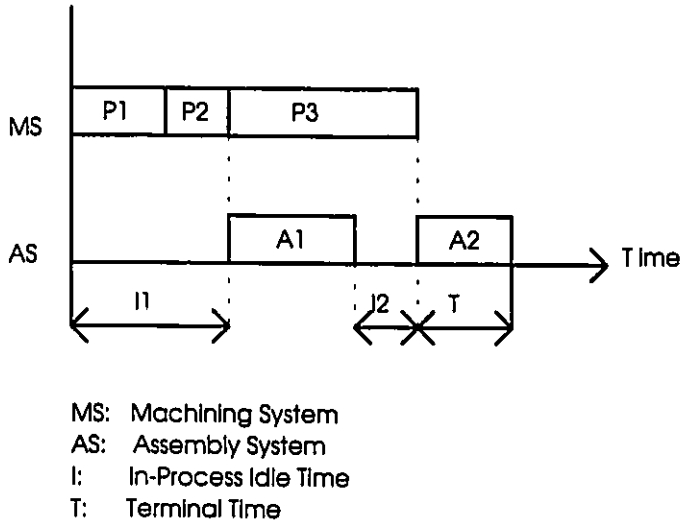


Figure 2.7 A possible final schedule for the example product *P*

For most problems, many different algorithms are available. In order to decide which one is the best implementation, there are some certain techniques describing the performance of fundamental algorithms regarding to their primary input parameters. However, comparing the algorithms can be challenging indeed, certain guidelines will be useful.

There are two classes of algorithms by the rate of growth of their complexity function. The first one has a complexity function which can be expressed in terms of a polynomial by given a problem instance of size n. They are called polynomial time algorithms. A function f is said to be of complexity $O(n^k)$, where k is a positive integer, if and only if there is a constant $c > 0$ such that

$$f(n) \leq cn^k$$

for all $n \geq n_0$, where n_0 is a positive integer that usually represents the smallest size of the problem instances involved.

The second class of algorithms consists of those whose complexity functions are not bounded by complexity $O(n^k)$ for some k. They are usually referred to as exponential time algorithms. The distinction is significant when considering large problem instances. Here the primary parameter n, usually the problem instance of size, which affects the running time most significantly. Most instructions of most programs are executed once or at most only a few times. If all the instructions of a program have this property, it is said that its running time is *constant*. When the running time of a program is *logarithmic* (i.e., log n), the program gets slightly slower as n grows. This commonly occurs in programs that solve a big problem by transforming it into smaller programs. *Linear* running time is achieved in the case that a small amount of processing is done on each input element.

Sometimes it is tried to solve a big problem by breaking it up into smaller subproblems. These subproblems are usually solved independently. At the end, all achieved solutions are combined. For those algorithms, $n \log n$ running time arises. Sometimes *quadratic* or *cubic* running times are achieved according to the processed data items (e.g., pairs of data causes quadratic running time, and triples of data items causes cubic running time etc.) (Sedgewick 1988).

In this thesis, detailed analysis of computational complexity of SIMEX is not discussed. The terminology of complexity is used as a measurement for the difficulties of identifying existing problem domain, in terms of entities and functional relationships among these entities.

3 CONVENTIONAL PRODUCTION SCHEDULING MODELS AND ALGORITHMS

In this section, the most popular production scheduling models and algorithms in the literature will be discussed under two main categories which are (1) Operation scheduling, (2) Product scheduling. The names of the production scheduling models and algorithms belonging to the first group are (1) Scheduling n operations on a single machine, (2) Two-machine flow shop problem, (3) Two-machine job shop problem, (4) A special case of a three-machine flow shop problem, and (5) M-machine flow shop model. The ones in the second group are (1) single-product scheduling problem and (2) N-product scheduling problem.

3.1 Operation Scheduling Models and Algorithms

For conventional systems a short definition of an algorithm is given in section 2.5. Algorithms differ from heuristics by their predictable behavior compared to heuristics which are mostly used in knowledge-based systems for their unpredictable behavior (Kusiak 1991). Kusiak also describes how to combine of heuristics and algorithms in solution of production scheduling problems. He called this combinations as heuristic algorithms (1991). In this chapter, only some conventional algorithms are given.

3.1.1 Preliminaries for Operation Scheduling

In general, preliminaries for production scheduling problems may be given as follows:

n denotes number of operation

o_i denotes operation i to be processed, where $i = 1, \ldots, n$

p_{ij} denotes processing time of operation i on machine M_j

r_i denotes readiness of operation o_i for processing

d_i denotes due date, i.e. promised completion time

There might be a set of precedence constraints among operations. Then, the processing of o_i must be completed before o_{i+1} can be represented as " $o_i < o_{i+1}$". It means that the operations are partially ordered by <.

For each operation o_i in a given schedule, the following parameters may be defined:

C_i denotes completion time of operations o_i

F_i denotes flow time, i.e. the difference between completion time and readiness, e.g.,

$F_i = C_i - r_i$

L_i denotes lateness, i.e. the difference between completion time and due date, e.g.,

$L_i = C_i - d_i$

T_i denotes tardiness, e.g.,

$T_i = \max \{C_i - d_i , 0\}$

In order to evaluate schedules (optimum, or how close to optimum), the following basic performance measures may be used.

- makespan: $C_{\max} = \max \{C_i\}$
- mean flow time: $F = 1/n \sum_{i=1}^{n} F_i$
- maximum lateness: $L_{\max} = \max \{L_i\}$
- mean tardiness: $T = 1/n \sum_{i=1}^{n} T_i$

3.1.2 Scheduling *n* Operations on a Single Machine

Two cases of this problem may be considered. (1) Due dates are not imposed, (2) Due dates are imposed. (Baker 1992, Kusiak 1990).

Case 1.

If one tries to minimize the total flow time, it is minimized by Shortest Processing Time (SPT) sequencing ($p_1 \leq p_2 \leq \ldots \leq p_n$).

Example 3.1

Determine the SPT schedule for a problem with seven operations for the following data:

Operation Number	1	2	3	4	5	6	7
Processing Time	2	1	1	3	5	3	1

The SPT schedule with respect to operation numbers is: {2, 3, 7}, 1, {4, 6}, 5. The operations shown in the curly braces can be selected arbitrarily without losing optimality, since they have identical processing times. I chose in the ascending order of operation numbers.

Case 2.

If due dates are imposed, the objective is to minimize the maximum tardiness or maximum lateness. They are minimized by Earliest Due Date (EDD) sequencing ($d_1 \leq d_2 \leq \ldots \leq d_n$).

Example 3.2

Determine the EDD schedule for a problem with seven operations for the following data:

Operation Number	1	2	3	4	5	6	7
Processing Time	2	1	1	3	5	3	1
Due Date	7	9	3	5	11	32	20

The EDD sequence with respect to operation numbers is : 3, 4, 1, 2, 5, 7, 6. The lateness and tardiness for each operation in the sequence are shown in the following table.

Operation Number	Due Date	Completion Time	Lateness	Tardiness
3	3	1	-2	0
4	5	4	-1	0
1	7	6	-1	0
2	9	7	-2	0
5	11	12	1	1
7	20	13	-7	0
6	32	16	-16	0

It is easy to note that operation 5 is late for one day. ($L_2 = 1$).

3.1.3 Two-Machine Flow Shop Problem

Kusiak (1986) developed an efficient implementation of the Johnson's Algorithm to solve the problem of scheduling n operation on two machines. Each operation can be performed on both machines with different processing times. Therefore, the objective is to minimize the maximum flow time. The algorithm is given according to the graphic scheme developed by Ören (1984) in Figure 3.1:

- Set $k = 1$
- Set $l = n$

For each operation

- Select the shortest processing time and
- Select corresponding machine number

End For

- Sort the list in increasing value of processing time including the triplets (operation number, processing time, machine number)

For each entry in the sorted list

If machine number is 1

then	• Set the operation sequence number to k • Set $k = k + 1$
else	• Set the operation sequence number to l • Set $l = l - 1$

End If

End For

Figure 3.1 Two-machine flow shop algorithm

Example 3.3

Solve the two-machine flow shop scheduling problem for the following set of data:

Operation Number i	Processing Time t_{ij} of Operation i on Machine j	
	j=1	j=2
1	4	2
2	3	3
3	7	6
4	2	3
5	8	9
6	2	1
7	4	3

After performing step 2, the following set of triples can be obtained;

Operation Number	$\min\{t_{i1}, t_{i2}\}$	Machine Number (j)
1	2	2
2	3	1
3	6	2
4	2	1
5	8	1
6	1	2
7	3	2

After performing step 3, the sorted set of triples are as follows:

Operation Number	$\min\{t_{i1}, t_{i2}\}$	Machine Number (j)
6	1	2
1	2	2
4	2	1
2	3	1
7	3	2
3	6	2
5	8	1

Step 4 produces the optimal sequence: 4, 2, 5, 3, 7, 1, 6.

3.1.4 Two-Machine Job Shop Problem

Kusiak (1990) described an algorithm, which uses Johnson's Algorithm in a heuristic fashion. First, he partitioned a set of n operations into the following types:

Type A. Operations to be processed only on machine 1

Type B. Operations to be processed only on machine 2

Type C. Operations to be processed on both machines in the order machine 1 and machine 2.

Type D. Operations to be processed on both machines in the order machine 2 and machine 1.

After partitioning the sets, the algorithm is given in Figure 3.2.

• Schedule the operations of type A in any order to obtain the sequence of S_A
• Schedule the operations of type B in any order to obtain the sequence of S_B
• Schedule the operations of type C according to the algorithm given in section 3.1.3 which produces the sequence of S_C
• Schedule the operations of type D according to the algorithm given in section 3.1.3 which produces the sequence of S_D
• Construct an optimal schedule in the form of the following Table 3.1.

Figure 3.2 Two-machine job shop algorithm

Table 3.1 Processing orders for machines 1 and 2

Machine	Processing Order
Machine 1	S_C, S_A, S_D
Machine 2	S_D, S_B, S_C

3.1.5 Special Cases of Three-Machine Flow Shop Problem

The two-machine flow shop problem is described by Kusiak (1990) to a special case of three machine flow shop problem with a condition found by French (1982). The condition is the maximum processing time on the second machine is no greater than the minimum time on either the first or the third machine. i.e.,

$$\text{either} \quad \min_{i=1}^{n}\{p_{i1}\} \geq \max_{i=1}^{n}\{p_{i2}\}$$

$$\text{or} \quad \min_{i=1}^{n}\{p_{i3}\} \geq \max_{i=1}^{n}\{p_{i2}\} \qquad (3.1)$$

where n denotes number of operations, and p_{ij} denotes the processing time of operation i at machine j. If the condition given above holds, then optimal sequencing of operations may be found by forming a pseudo two-machine problem

$$a_i = p_{i1} + p_{i2}$$
$$b_i = p_{i3} + p_{i2}$$

and by using the algorithm given in section 3.1.3 with these a_i and b_i values. This solution is called Johnson's approximate method. There are some other cases shown by Burns and Rooker (1976), and Szwarc (1977). In these cases, it is possible to find an

optimum without enumerative search. Some of the cases may be counted as follows similar to the shown above:

Case 1. If machine 1 dominates machine 2:

$$\min_{i=1}^{n}\{p_{i1}\} \geq \max_{i=1}^{n}\{p_{i2}\}$$

then use Johnson's approximate algorithm described above.

Case 2. If machine 3 dominates machine 2:

$$\min_{i=1}^{n}\{p_{i3}\} \geq \max_{i=1}^{n}\{p_{i2}\}$$

then use Johnson's approximate algorithm.

Case 3. If $p_{i2} \leq \min\{p_{i1}, p_{i3}\}$ for all i, then use Johnson's approximate algorithm.

Example 3.4

Given the data, solve the three-machine flow shop scheduling problem.

	Actual Processing Time			Constructed Processing Times	
Operation	p_{i1} M_1	p_{i2} M_2	p_{i3} M_3	a_i First M/C	b_i Second M/C
1	5	2	6	7	8
2	3	3	2	6	5
3	7	2	9	9	11
4	4	1	4	5	5
5	5	1	1	6	2
6	3	2	5	5	7

Let us check whether (3.1) holds for this problem, or not.

1) $\min_{i=1}^{6}\{p_{i1}\} = 3 \geq \max_{i=1}^{6}\{p_{i2}\} = 3$, which holds

2) $\min_{i=1}^{6}\{p_{i3}\} = 1 \geq \max_{i=1}^{6}\{p_{i2}\} = 3$, which does not hold.

Only one condition is enough for (3.1), therefore algorithm discussed in section 3.1.3 can be applied.

Operation Number i	Processing Time t_{ij} of Operation i on Machine j	
	j=1	j=2
1	7	8
2	6	5
3	9	11
4	5	5
5	6	2
6	5	7

After performing step 2, the following set of triples can be obtained;

Operation Number	$\min\{t_{i1}, t_{i2}\}$	Machine Number (j)
1	7	1
2	5	2
3	9	1
4	5	1
5	2	2
6	5	1

After performing step 3, the sorted set of triples are as follows:

Operation Number	$\min\{t_{i1}, t_{i2}\}$	Machine Number (j)
5	2	2
2	5	2
4	5	1
6	5	1
1	7	1
3	9	1

Step 4 produces the optimal sequence: 4, 6, 1, 3, 2, 5.

3.1.6 *M*-Machine Flow Shop Model

Baker (1992) described a production scheduling problem for *m*-machines with *n* jobs. In this model, interprocess queues are not allowed. Once the processing of a job begins, subsequent processing must be carried out with no delays in the operation sequence. He considers the problem of minimizing makespan with an optimal permutation schedule when no interprocess queues are allowed.

Suppose that jobs *i* and *j* are adjacent in sequence and that job *i* precedes job *j*. A certain delay would be incurred in the processing of job *j* if the two jobs were released to the shop at the same time and job *i* processed first. Let I_{hj} denote the idle time or delay incurred by job *j* prior to its operation on machine *h*. Now suppose that job *j* is followed in sequence by job *k*. The delays incurred in the processing of job *k* do not depend on what happened before job *j* in sequence, but only operation times of job *j* itself, when it is processed without delay. Let D_{ij} denote the total delay incurred by job *j* when it follows job *i* in sequence, measured from the start of job *i*. i.e.,

$$D_{ij} = I_{1j} + I_{2j} + \text{L} + I_{mj}$$

Similarly define D_{jk} to be

$$D_{jk} = I_{1k} + I_{2k} + \text{L} + I_{mk}$$

which is the total delay incurred by job k when it follows job j in sequence, measured from the start of job j. If there are only three jobs, the expression for the makespan of the schedule associated with the sequence i-j-k is

$$M = D_{ij} + D_{jk} + p_{1k} + p_{2k} + p_{3k}$$

The general expression for the makespan can be given as

$$M = \sum_{j=1}^{n-1} D_{[j],[j+1]} + \sum_{h=1}^{m} p_{h,[n]}$$

Therefore, the makespan is the sum of two quantities: (1) sum of sequence dependent delay terms and (2) total processing time of the last job in the sequence. Here, there is an analogy between the traveling salesperson problem (TSP) and the problem described above. Each city corresponds to a job, and the intercity distances correspond to the delay pairs D_{ij} with a dummy city corresponding the idle state to which the distance from one city i is the sum of the operation times for job i, and from which the distance to all other cities are zero. i.e. look at the Figure 3.3,

from \ to	1	2	3	...	n	$n+1$
1	-	D_{12}	D_{13}	...	D_{1n}	$\Sigma_{h=1}^{m} P_{h,1}$
2	D_{21}	-	D_{23}	...	D_{2n}	$\Sigma_{h=1}^{m} P_{h,2}$
...	...	...	...	...	...	...
n	D_{n1}	D_{n2}	D_{n3}	...	-	$\Sigma_{h=1}^{m} P_{h,n}$
$n+1$	0	0	0	0	0	-

Figure 3.3 Distance matrix of the cities.

Example 3.5

As an example, suppose the problem contains four jobs and five machines with the processing times given below.

job j	1	2	3	4
p_{1i}	4	2	5	3
p_{2i}	5	4	3	9
p_{3i}	7	6	2	9
p_{4i}	9	8	1	8
p_{5i}	3	7	6	4

And then it is possible to create the TSP distance matrix as

jobs	1	2	3^0	4	Dummy
job 1	-	13^3	17	9	28
job 2	4	-	16	3	27^2
job 3	18	6	-	5	17
job 4	13^4	17	22	-	33
Dummy	0	0	0^1	0	-

For verification, the first cell, D_{12}, is calculated on a Gantt chart (Figure 3.4).

For the solution technique, we can use nearest neighbor heuristic (NNH). Let us begin with job 3 as a origin. Then we can go to the nearest neighbor (shortest time), which is Dummy, among the neighbors available {1, 2, 4, Dummy}. From Dummy, we can go to the nearest neighbor among {1, 2, 4}, which is 2. From job 2, we can go to the nearest neighbor among {1, 4}, which is 1. And the last one is job 4. So we will have a sequence 3-Dummy-2-1-4 with a makespan of 53. In order to obtain the best sequence, we have to start from every job i as origin, apply the NNH, and then select the sequence with the minimum makespan.

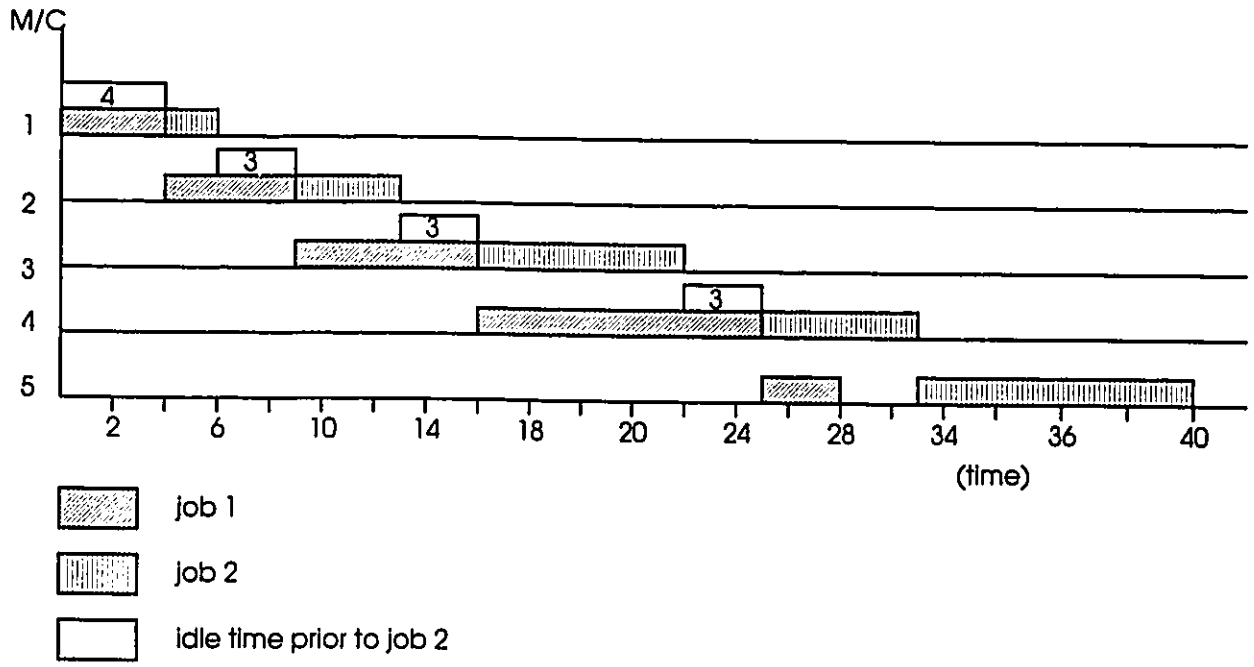


Figure 3.4 Gantt chart for the scheduling of Job 1 and Job 2 on 5 machines. Total idle time prior to Job 2 is summation of the idle times shown on the figure, which is 13.

3.2 Product Scheduling Models and Algorithms

3.2.1 Preliminaries for Product Scheduling

Consider a digraph representation of the product which consists of a number of parts, *P*, and subassemblies, *A*. In the digraph, each node is labeled as (a, b, c), where a is the machining time, b is the subassembly time, c is the level of depth, i.e., depth measured between the root node versus the initial nodes (Figure 3.5). In this digraph, any node of degree 1, i.e., with the number of edges incident to the node equal to 1, denotes a part; and any node of degree greater than 1 denotes a subassembly or a final product.

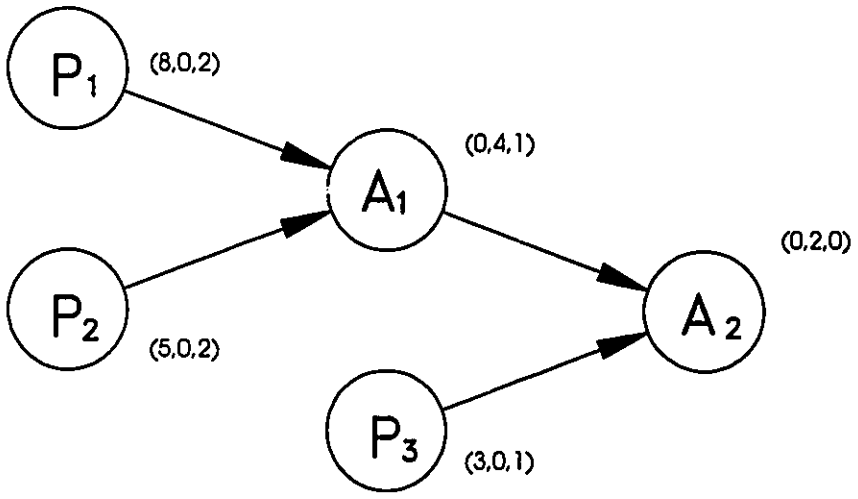


Figure 3.5 A digraph of the example product *P*

A complex digraph can be decomposed into simple subdigraphs by removing a number of nodes corresponding to the final assembly or subassemblies. A simple digraph is a digraph in which each node of a degree greater than one has at most one preceding node of a degree greater than one (Figure 3.6). A complex digraph consists of more than one simple digraph (Figure 3.7).

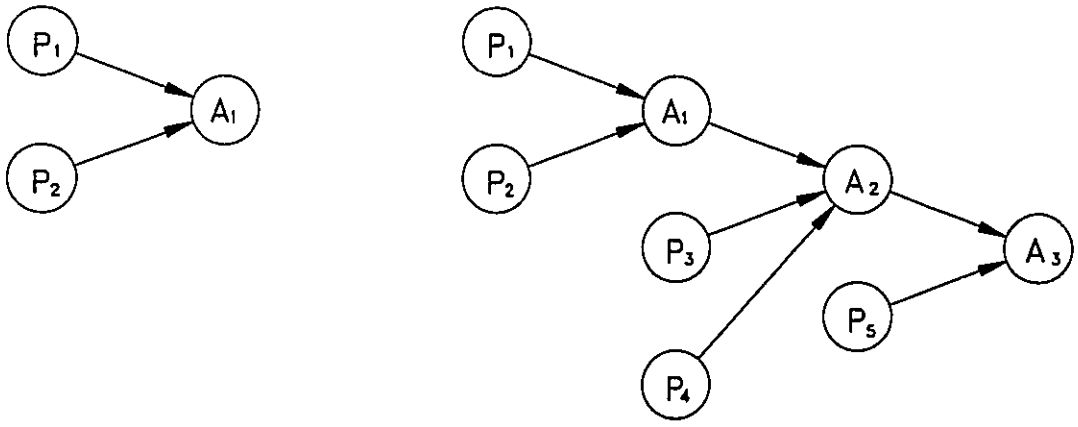


Figure 3.6 Two examples of simple digraphs

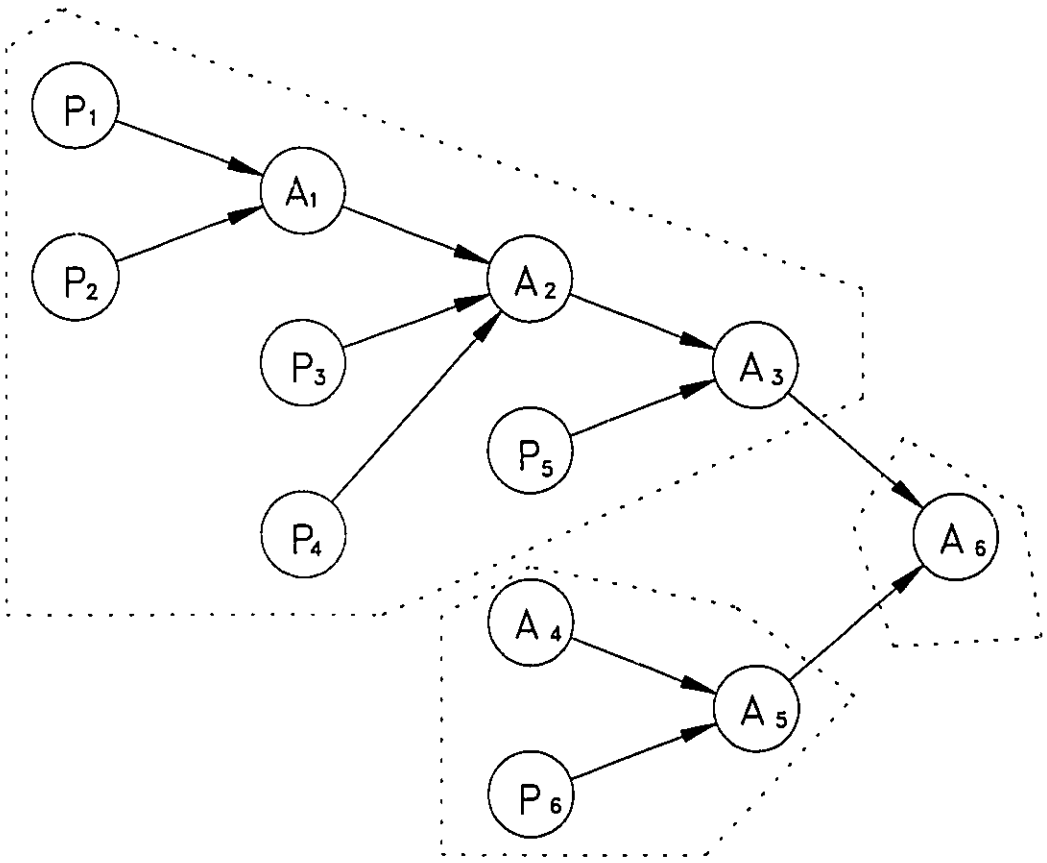


Figure 3.7 An example of a complex digraph. Dot lines encapsulates the simple digraphs.

Theorem 3.1

Scheduling nodes of a simple digraph G_s with the maximum level of depth first (MLDF) provides the minimum makespan schedule. The proof of the theorem is due to Kurisu (1976).

Theorem 3.2

Consider a subassembly or a final product P represented by a complex digraph G and decompose it into subdigraphs $g_1, g_2, ..., g_t$ by removing the root node v_0 of G. Let $S(g_i)$ be the minimum makespan partial schedule associated with g_i, $i = 1, ..., t$. If parts and subassemblies corresponding to g_i and g_j, $i \neq j$ are preempted, then the minimum makespan schedule of product P is as follows:

$$S(P) = \{S_1(G), S_2(G), v_0\}$$

where

$$S_1(G) = [S(g_{[1]}), S(g_{[2]}), ..., S(g_{[k]})], \text{ for } I_{[i]} \leq T_{[i]}, i = 1, ..., k$$

is a schedule obtained using the longest in-process idle time last (LITL) rule and

$$S_2(G) = [S(g_{[k+1]}), S(g_{[k+2]}), ..., S(g_{[t]})], \text{ for } I_{[i]} > T_{[i]}, i = k+1, ..., t$$

is a schedule obtained using the longest terminal time last (LTTL) rule. The proof of the theorem is due to Kurisu (1976).

3.2.2 Single Product Scheduling Problem

Based on the theorems 3.1 and 3.2, the algorithm for single product scheduling (SPS) developed by Kusiak (1991) and expressed according to the scheme of Ören (1984) is given in Figures 3.6a and 3.6b.

Figure 3.8a Algorithm for Single Product Scheduling Problem (SPS) (Part 1)

<table>
<tr><td colspan="2">• Label all nodes of the digraph G representing the structure of the product considered</td></tr>
<tr><td colspan="2">If G is a simple digraph</td></tr>
<tr><td>then</td><td>• Use the MLDF rule to generate optimal schedule of product P
• Stop</td></tr>
<tr><td colspan="2">End If</td></tr>
<tr><td colspan="2">• Remove root node v_0 form G
• Decompose it into subdigraphs g_l, $l = 1,..., L$</td></tr>
<tr><td colspan="2">If all g_l are simple digraphs</td></tr>
<tr><td>then</td><td>• Set $k = 0$</td></tr>
<tr><td>else</td><td>• Decompose each g_l which is not a simple digraph by removing its root node
• Let v_j denote a root node which has been removed, $j = 1, ..., J$
• Set $k = j$</td></tr>
<tr><td colspan="2">End If</td></tr>
</table>

Continued on the next page ...

Figure 3.8b Continuation of Algorithm Single Product Scheduling (Part 2)

While k > 0

- Let g_{ik} denote the simple subdigraphs associated with v_k
- Use MLDF rule to generate the minimum makespan partial schedule $S(g_{ik})$ for each subdigraph g_{ik}, $i = 1, ..., N_k$, where N_k is number of subdigraphs obtained after v_k has been removed

For each partial schedule $S(g_{ik})$

- Determine the in-process idle-time I_{ik}
- Determine the terminal time T_{ik}, $i = 1, ..., N_k$

End For

- Separate $S(g_{ik})$ into two lists:

  List 1: schedules $S(g_{ik})$ such that $I_{ik} \leq T_{ik}$

  List 2: schedules $S(g_{ik})$ such that $I_{ik} > T_{ik}$

- Use LITL rule to generate

  $S_1(g_k) = [S(g_{[1]k}), S(g_{[2]k}), ..., S(g_{[r]k})]$, for $I_{[i]} \leq T_{[i]}$, $i = 1, ..., r$

- Use LTTF rule to generate

  $S_2(g_k) = [S(g_{[r+1]k}), S(g_{[r+2]k}), ..., S(g_{[t]k})]$, for $I_{[i]} > T_{[i]}$,

  $i = r+1, ..., t$, $t=N_k$

- Generate the partial schedule

  $S(g_k) = \{S_1(g_k), S_2(g_k), v_k\}$

If $v_k = v_0$

then $S(P) = S(g_0)$ is the optimal schedule. Stop.

End If

- Consider $S(g_k)$ as a simple subdigraph schedule
- Calculate I_{ik} and T_{ik}. Set $k = k - 1$

End While

3.2.3 *N*-Product Scheduling Problem

Solution for *N*-product scheduling problem (NPS) is given by Kusiak (1991). He assumes that *N*-product scheduling problem can be decomposed into *N*-single product scheduling problem by product-by-product policy (e.g., each product can be scheduled by single product scheduling algorithm independently). The algorithm for scheduling of N products $P_1, ..., P_N$ is presented in Figure 3.7.

For each product P_i, $i = 1,..., N$

- Determine the optimal schedule for $S(P_i)$ using algorithm SPS

End For

For $i = 1,..., N$

- Separate $S(P_i)$ all into the following two lists:

  List 1: including $S(P_i)$ such that $I_i \leq T_i$, $i = 1, ..., k$

  List 2: including $S(P_i)$ such that $I_i > T_i$, $i = k+1, ..., N$

End For

- Develop the LITL schedule for the schedules in List 1, e.g.,

$$S_1(NP) = \{S(P_{[1]}), S(P_{[2]}), ..., S(P_{[k]})\}$$

- Develop the LTTF schedule for the schedules in List 2, e.g.,

$$S_2(NP) = \{S(P_{[k+1]}), S(P_{[k+2]}), ..., S(P_{[N]})\}$$

- Generate final schedule, e.g.,

$$S(NP) = \{S_1(NP), S_2(NP)\}$$

Figure 3.9 Algorithm NPS

4 KNOWLEDGE-BASED PRODUCTION SCHEDULING SYSTEMS

In this section, basic concepts of knowledge-based systems and basic components knowledge-based production scheduling systems are discussed with the advantages regarding to the conventional programming. In addition, an overview of knowledge-based production scheduling systems is given with recent developments, and some of major manufacturing knowledge representation schemes are introduced with the examples from literature.

4.1 Basic Concepts of Knowledge-based Systems

In recent years, a number of important subjects emerged about computer aided problem solving techniques in which knowledge-based system developers are currently involved due to the advantages compare to traditional programming techniques. These advantages are: (1) the use of computer to do symbolic reasoning, (2) applications of heuristic solution techniques for the problems that do not respond to algorithmic solution strategies, (3) a concern with problem solving using inexact, missing, or poorly defined information, (4) capturing and manipulating the significant qualitative features of a situation rather than relying on numeric methods, (5) dealing with issues of semantic meaning as well as syntactic form, (6) finding answers that are neither exact nor optimal, but are "sufficient" with the use of heuristic solution techniques, (7) the use of large amount of domain-specific knowledge, (8) meta-level languages to have more sophisticated control of problem-solving strategies, and (9) learning behavior from the current events not to make same kinds of solutions again and again, and respond quickly.

Knowledge-based systems are constructed using the techniques of declarative knowledge representation. The purpose of a knowledge-based system is to assist decision makers,

within a domain of application. The performance of a knowledge-based system can match or even exceed that of humans by incorporating knowledge from diverse sources.

A knowledge-based system can be constructed in a modular fashion which facilitates a partition of the program. The partitioning of the program into comprehensible modules facilitates the process of functional modifications as well as incorporation of different technologies. Otherwise, the system remains immutable, stagnates and then loses its value over time due to changing environmental circumstances. A modification of a program through some addition of capabilities may imply an increase in system intelligence. Even when the repertoire of functions remains static, intelligence may be enhanced through increased effectiveness. A straightforward way to implement such changes is for the programmer to update the knowledge base accordingly. Even the results of current research toward machine learning indicate that self-programming capabilities will become feasible in the near future.

In the commercial world, expert systems are well known knowledge-based systems. However, not every knowledge-based system can be considered an expert system. An expert system is capable of explaining its behavior and its decisions to the user, as human experts do. By a user-friendly interaction, it will make the system's reasoning transparent to the user. Expert systems also have the ability to deal with uncertainty and incompleteness. Information about the problem to be solved can be incomplete or unreliable; relations in the problem domain can be approximate.

In order to build a knowledge-based system, in general, both the functions of problem solving and user-interface must be developed. The problem solving function must be capable of using domain-specific knowledge, and the user-interface function must simplify the life for the user to communicate with the system.

Knowledge-based systems, in general, consist of three main modules: (1) a knowledge base, (2) an inference engine, (3) a user interface as displayed in Figure 4.1.

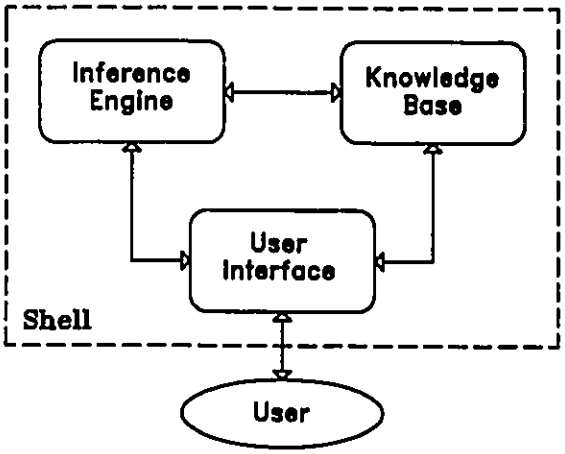


Figure 4.1 Structure of a knowledge-based system

A knowledge base consists of the knowledge that is specific to the domain of application (e.g., simple facts about the domain, production rules that describe relations or phenomena in the domain, methods like heuristics and ideas for solving problems in the domain, etc.). An inference engine is designed to use the knowledge in the base actively. An inference engine is a set of reasoning procedures that acts on the domain knowledge to respond to the queries. The reasoning procedures that implement the control of knowledge base are separated from the production rules themselves.

This separation is important for several reasons: (1) this separation makes it possible to represent knowledge in a more natural fashion. For example, *if ...then* rules are closer to the way in which human beings describe their own techniques rather than a program that

embeds this knowledge in lower-level computer code, (2) a knowledge-based builder can focus directly on capturing and organizing the problem-solving knowledge rather than on the details of its lower-level computer implementation, (3) separation of control and knowledge, along with the modularity of rules and other knowledge representation schemes, allows changes to be made in one part of the knowledge base without creating side effects in other sections of the program code, (4) separation of knowledge base allows to use interface software to communicate with a naive user.

A user interface provides a smooth communication between the user and the system, so that the user can utilize the system with no knowledge of the details of implementation. It makes access more comfortable for the human and hides much of the system complexity.

4.2 Knowledge-based Production Scheduling Systems

The recent development of computer aided solution methods affected all engineering areas including manufacturing engineering. The use of artificial intelligence techniques to solve production scheduling problems is a new trend for researchers. There are several reasons behind this trend: (1) traditional programming techniques can help only conventional production scheduling models (2) conventional production scheduling models underestimate the complexity of the manufacturing environment constraints, (3) conventional production scheduling models cannot realistically represent unexpected events in a dynamic manufacturing environment such as machine breakdowns, material shortages and so forth, (4) some production scheduling problems do not respond to algorithmic solutions, (5) it is often difficult to apply heuristic approaches with the traditional programming techniques, and sometimes it is impossible. In general, conventional scheduling techniques are based on many assumptions to make the problem solvable in traditional ways which deals with the scheduling problems in an unrealistic manner.

The knowledge-based production scheduling systems are promising to overcome all these problems listed above with the help of artificial intelligence techniques. The properties of a knowledge-based production scheduling system may be described as follows:

- Learning behavior can be possessed with generalization of a current problem, and storing the generalized problem in learning base for future use for, or updating the knowledge base in real time.
- Complex problems that do not respond to conventional algorithmic solutions can be solved with the help of heuristic approaches, with numerous options and factors. Because, knowledge-based production scheduling systems allow to use heuristics, and rule-of-thumbs which are the simplifications that help limit the search for possible problem solutions.
- Symbolic processing, object-oriented software engineering techniques can be applied. Instead of algorithms, symbols can be used to represent real world objects, and characteristics associated with them. Moreover, complex relations among the objects can be easily modeled.
- Time events can also be represented as objects, therefore dynamic plant behavior can be easily simulated.
- The models can be continuously updated to keep them current.

For the rest of this section, I will introduce the structure of general knowledge-based production scheduling systems.

The basic components of a knowledge-based production scheduling system are shown in Figure 4.2. As shown in Figure 4.2, knowledge-based production scheduling systems consist of five main components: (1) knowledge base, (2) algorithm base, (3) data base, (4) inference engine, (5) user interface.

In a *knowledge base and data base,* description of scheduling problems, parts, operations, and resources (e.g., machines, tools, and material etc.), in other words, manufacturing knowledge are declared with the help of knowledge representation schemes (e.g., first-order logic, production rules, frames, etc.). Some of major knowledge representation schemes will be discussed in detail in next section with examples from the literature. The distinction between the knowledge base and the data base is that the knowledge base contains the problem solving knowledge of a particular application. For example, in a rule based system, this knowledge is represented in the form of *if...then* production rules.

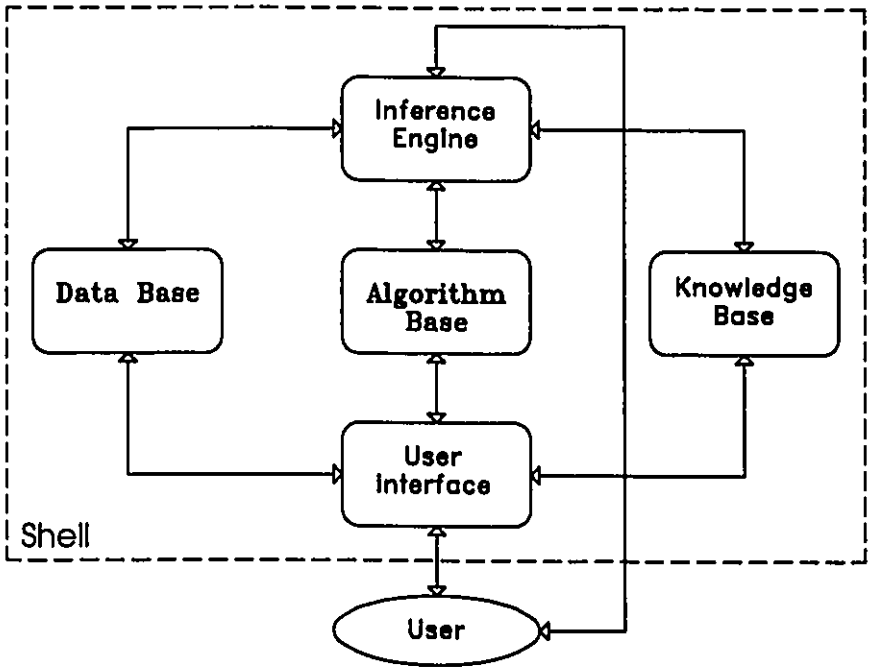


Figure 4.2 Structure of a knowledge-based production scheduling system

In an *algorithm base*, scheduling algorithms and heuristics (e.g., single machine scheduling, two-machine flow shop problem and so forth) can be stored. These

algorithms or heuristics are selected based upon the characteristics of the scheduling problem stored in knowledge base.

The *inference engine* controls the procedures of triggering rules in the knowledge base and algorithm base. It is the interpreter of the knowledge base. The procedures that implement the control cycle are separate from the production rules themselves. This separation is important for several reasons: (1) allows the possible representation of manufacturing knowledge in a more natural fashion. For example, *if ...then* rules are closer to the way in which human beings describe their own techniques rather than a program that embeds this knowledge in lower-level computer code, (2) a knowledge-based builder can focus directly on capturing and organizing the production scheduling problem-solving knowledge rather than on the details of its lower-level computer implementation, (3) separation of control and knowledge, along with the modularity of rules and other knowledge representation schemes, allows changes to be made in one part of the knowledge base without creating side effects in other sections of the program code, (4) separation of knowledge base eases the use of the interface software to communicate with a user.

A *user interface* provides a smooth communication between the user and the system, so that the user can utilize the system with no knowledge of the details of implementation. It makes access more comfortable for the human and hides much of the system complexity.

4.3 An Overview of Knowledge-based Production Scheduling Systems

Developments in artificial intelligence have an impact on manufacturing systems. Almost all areas of modern manufacturing have been affected by artificial intelligence. Researchers and engineers are developing intelligent systems ranging from applications in

machining and assembly to production planning and scheduling under the subject of *Intelligent Manufacturing Engineering* (IME).

Beyond the traditional methods, IME deals with important current issues in production scheduling, e.g., knowledge-based manufacturing system design and development. To date, numerous papers have been published on this subject.

Fox and Smith (1984) describe an Intelligent Scheduling and Information System (ISIS) developed at the Carnegie-Mellon University for job shop scheduling. This system contains all the knowledge necessary to plan and schedule production in a job shop environment. It takes into consideration all relevant constraints in the construction of job schedules (Badiru 1990). The system selects a sequence of operations needed to complete an order, determines start and end times, and assigns resources to each operation. It can also act as an intelligent assistant, using its expertise to help schedulers maintain schedule consistency and identify decisions that result in unsatisfied constraints. Knowledge in the system includes organizational goals, such as due dates; physical constraints, such as limitations of particular machines; and causal constraints, such as the order in which operations must be performed.

Another knowledge-based job shop scheduling system, called MASCOT, has been developed by Erschler and Esquirol (1986). The system attempts to make resources (machines) constantly available and to finish jobs before the required due dates. The system was applied to solve a scheduling problem involving machines and operations only. Start times of operations and constraints that use common resources are considered as important aspects of the problem. The constraint-based analysis approach is used to generate precedence relationships from conflicting resources. Consequently, only two

types of production rules were established in the knowledge base: time updating rules and sequencing rules to generate precedence among operations (Kusiak 1991).

Smith (1988) describes a constraint-based framework for reactive management of factory schedules. The framework is implemented in a knowledge-based scheduling system called OPIS. He discusses the concept of *predictive scheduling* which is the technique for establishing a priori guidance in coordinating activities on the factory floor with a given set of production objectives, a state of the factory at the point that schedule is called for, and a predictive model of how the factory will operate over a specified time horizon. He states that these kinds of traditional approaches to scheduling only simplifies the problem to manage the complexity, and to produce schedules that often have very little relationship to actual unpredictable factory conditions. The factory floor is a very dynamic environment. A lot of unexpected events can occur during the manufacturing (i.e., machines break down, raw materials fail to arrive on time, rework required by quality control, and so on). In order to increase the effectiveness of the schedules, predictive schedules must be managed with the unexpected events reactively. For that reason, he developed the framework for reactive schedule management to handle the unexpected events.

Miyashita and Sycara (1992) developed a Case-based Interactive Scheduler (CABINS) whose goal is to support interactive schedule repair. The technique that they used is called Case-based Reasoning (CBR) which is a recent AI problem solving paradigm (Kolodner, and et al, 1985). A CBR system tries to solve a problem by retrieving the most similar case with the current problem from its case base, modifying it to adapt to the current situation and applying it to the current problem. After having the solution, the solved problem is stored as a new case in the case base. In the case base, the unit of knowledge is a case, which is an experience encountered during previous problem solving

activities. This makes it easier to examine and evaluate the knowledge. A CBR system can learn. It can remember its performance and modify its behavior to avoid repeating prior mistakes. It is also an adaptive system because by reasoning from analogy with the past experiences, it can construct solutions to different problems.

The CABINS is an interactive schedule repair. To the user initial schedule is produced by an automated scheduling system. After the initial schedule is made, it is examined by the user and the defect detector (a rule-based system module of the CABINS) to find undesirable parts in the existing schedule. If some defects are monitored, the information about the defects are passed to the repairer which is another module of the CABINS. Resource allocations in the current schedule is modified or canceled by the repairer module, and the scheduler module is asked to re-schedule the conflicting operations whose resource allocations are canceled. The overall goal of CABINS is to make repairs as cheap as possible trying at the same time to minimize interfering side effects of the repairs.

Sadeh (1992) developed a micro-opportunistic factory scheduler called Micro-Bottleneck Scheduling System (MICRO-BOSS). He renames the *dynamic scheduling* to *opportunistic scheduling*, where dynamic scheduling is allocation of manufacturing resources to perform certain tasks in a dynamic factory environment to reduce tardiness and inventory costs. Because the schedulers reexamine the problem each time a resource or a job has been scheduled. Whenever an opportunity arises they can reserve the resource or assign the job to the resource. However, opportunism is limited as they required scheduling large resource or job subproblems. For this reason, he calls these approaches as macro-opportunistic techniques.

Bottlenecks are the most dynamic resource elements in factory environment. They do not necessarily span over the entire scheduling horizon. In addition they tend to shift before being completely scheduled. A macro-opportunistic scheduler will not be able to consider that. Usually he will overconstrain his set of alternatives before having worked on the subproblems that will most critically determine the quality of the entire schedule. As a result, he will have poorer solutions. A more flexible approach had been investigated by Sadeh. It is called *micro-opportunistic* approach, in which the evolution of bottlenecks is continuously monitored during the construction of the schedule, and the problem solving effort regularly redirected towards the most serious bottleneck.

MICRO-BOSS is an implementation of micro-opportunistic approach. Each operation is considered to be an independent decision point. Any operation can be scheduled in any time, if appears appropriate to the scheduler. Even within the same job, upstream or downstream operations may not be simultaneously scheduled. It selects iteratively one operation to be scheduled, and then assigns start time of the operation. Whenever an operation is scheduled, a new search state is created, where new constraints are added to account for the reservation assigned to the operation. A consistency enforcing procedure is applied to every state to determine the remaining possible reservations of each unscheduled operation. If there is no reservation available for an unscheduled operation, a *dead-end state* has been reached., which means the system needs backtracking to undo some earlier reservation assignments to be able to complete the schedule. If the search state does not appear to be a dead-end, the scheduler continues to schedule a new operation by assigning a reservation to that operation. The opportunism in MICRO-BOSS comes from its ability to constantly revise its search strategy. Moreover, it can redirect the effort towards the scheduling of the operation that appears to be the most critical in the current search state.

4.4 An Overview of Manufacturing Knowledge Representation in Knowledge-based Systems

The key success of knowledge-based systems is the effective representation of domain-specific knowledge (De, 1988). Domain knowledge has many forms, including descriptive definitions of domain-specific terms; descriptions of individual objects, classes of objects, and their interrelationships; and criteria for making decisions. In many application domains like manufacturing, much of the information to be encoded into knowledge base originates from descriptive statements which are difficult and unnatural to represent by simple structures like arrays or set of numbers. In this section, we are going to review of knowledge representation schemes such as first-order logic, production rules, semantic networks and frames.

First-Order Logic (FOL)

FOL deals with the relationship of implication between assumptions and conclusions. It is not concerned with the truth, falsity, or acceptability of individual sentences, but with relationships among them. FOL sentences can be expressed as a collection of clauses. A clause is an expression of the following form:

$$B_1, ..., B_m \leftarrow A_1, ..., A_n , \; m, n \geq 0$$

where $A_1, ..., A_n$ are conditions of the clause and $B_1, ..., B_m$ are alternative conclusions of the clause. Both conclusions and conditions are expressions of the form $P(t_1,...,t_k)$ called an atom, where P is a k-argument predicate symbol, and $t_1,...,t_k$ are terms (De, 1988). A term may be in types of a variable, a constant symbol, or another predicate symbol.

If the clause contains the variables $\chi_1, ..., \chi_k$, then it can be interpreted as stating that

$$\forall \chi , \quad B_1, \text{ or } \ldots \text{ or, } B_m \text{ if } A_1, \text{ and } \ldots \text{ and, } A_n$$

In n is equal to zero, then the clause will be an unconditional assertion and interpreted as $\forall \chi , B_1$, or ... or, B_m. If m is equal to zero, the implication without conclusion will be a denial, $\forall \chi , A_1$, and ... and, A_n. If m and n are both equal to zero, then the clause can be interpreted as a sentence that is always false.

A variety of knowledge can be expressed using the clausal form of logic described above. Simple assertions can be expressed by clauses as follows:

IDLE (M_1, t) ←	(Machine M_1 is idle at time t)
MACH_OP (M_3,O_5) ←	(Machine M_3 can perform operation O_5)

In Prolog, the same assertions may be expressed as follows:

```
idle(machine_1, t) :- true.                  % Machine M1 is idle at time t
mach_op(machine_3, operation_5) :- true.     % Machine M3 can perform operation O5
```

Clauses can also express general conditional propositions such as

~ IDLE (M_1, t) ← MACH_PROC (M_1,x, y, t) (if machine M_1 is performing an operation x on a part y at time t, then M_1 is not idle at time t; denotes negation)

In Prolog,

```
mach_proc(machine_1, OpNo, PartNo, T) :- idle(machine_1, T).
```

A clause can have several joint conditions or several joint conclusions as well.

Horn Clause Subset of First-Order Logic

Many first order logic applications have found it sufficient to restrict the form of clauses to those containing at most one conclusion called Horn clauses. In fact, it can be shown

that any problem that can be expressed in first order logic can be reexpressed by means of Horn clauses (Kowalski, 1979). Syntactically, a Horn clause is an expression of the form

$$B \leftarrow A_1, ..., A_n$$

where B and A_i , $n \geq 0$ are atoms. It can be interpreted as stating that for all values of the variables it contains, B holds if A_1, ..., A_n hold. It is interesting to note that, Horn clauses can be interpreted as procedures where B is the procedure name, and all of A_1, ..., A_n is the procedure body.

Production Rules

These are often called productions. A production is a condition-action pair which defines a single chunk of problem-solving knowledge. The condition part of the rule is a pattern that determines when the rule may be applied to a problem instance. The action part defines the associated problem-solving step (Brownston et al. 1985).
Production rules are also called the language of if-then rules which is by far the most popular scheme for representing knowledge (Bratko, 1991). In general, such rules are conditional statements, but they can have various interpretations. Examples are

- *if* precondition **P** *then* conclusion **C**
- *if* situation **S** *then* action **A**
- *if* conditions **C1** and **C2** *then* condition **C** does not hold, etc.

Production rules usually turn out to be natural form of expressing knowledge, and have very useful additional features such as modularity: each rule defines a small, relatively independent piece of knowledge; incrementability: new rules can be added to the knowledge base relatively independent of other rules (i.e. without worrying about how

they will fit in, essentially important when the system can learn from the new knowledge or past experience); modifiability: old rules can be changed independently of other rules (i.e. without breaking the entire system); and transparency: production rules support system's transparency. Supporting the system's transparency is an important and distinguishing feature of expert systems. The transparency of the system means ability of the system to explain its decisions and solutions (Bratko, 1991). Production rules can facilitate answering basic type of user's questions (e.g. *how* and *why*).

As an alternative of production rules, structured production rules are suggested by Niwa et al. (1984). They described that the unstructured production systems have some certain disadvantages. One of them is the inefficiency of program execution due to the necessity of scanning a large number of production rules. In order to increase the efficiency, the rules can be structured by grouping into classes. Therefore, during inferencing, depending upon the context, the scanning can be performed on the required class of rules (e.g. in manufacturing, machining and shipping operations can be group into two different classes).

Although production rules have sufficient expressive power to represent a useful range of domain-dependent inference rules and behavior specifications, they are inadequate as a knowledge representation scheme in describing objects and static relationships, which is where frames are particularly effective (Kusiak, 1990).

Frames

We can organize knowledge into more complex units that represent complex situations or objects in an application domain. These units are called frames. The frame theory is suggested by Minsky (1975).

"Here is the essence of the frame theory: When one encounters a new situation (or makes a substantial change in one's view of a problem) one selects from memory a structure called a "frame." This is a remembered framework to be adapted to fit reality by changing details as necessary" (Minsky, 1975).

A frame provides a concise structural representation of relations. It can be thought as a generalization of a property list that provides a structured representation of an object or a class of objects. Each individual object or class of objects may be represented by a frame (Kusiak, 1990). In frame scheme, each property is called *slot*, and its particular instantiation is called a *value* or an *entry*. The most important features of a frame may be counted as follows:

- It provides a structured representation of objects and classes of objects in an application domain.
- It provides a mechanism (called inheritance) that guides description movement from class descriptions to individual descriptions. We may achieve this mechanism by building relationships among frames.
- It allows the specification of procedures (called demons) for computing descriptions, i.e. procedural attainment (Figure 4.3).
- Frame default information can be used for certain properties such as a car has four wheels (Figure 4.3)
- It allows to determine descriptions in the absence of specific knowledge. Frames can represent not only objects being reasoned about, but rules as well. Each rule represented as a frame can be grouped into classes and the description of the rule can include arbitrary attributes of the rule.
- A frame-based representation of domain-specific knowledge in an application domain and structural knowledge associated with modeling techniques can be used to

formulate analytical models that could solve various subproblems (Binbasioglu and Jarke, 1986; Murphy and Stohr, 1986).

Figure 4.3 illustrates the notion of a frame. It encodes the knowledge of a sports car, Panther with a convertible roof and two doors (Kim, 1991). A slot can have default values that pertain unless explicitly overridden; for example, the value of "(4)" for the number of wheels. A slot can contain a pointer referring to another frame, as illustrated by the engine model that refers to the V6-013 frame. Another type of entry is the demon procedure which is a program module that calculates appropriate value when invoked, based on other information. For example the trunk capacity can be obtained as the product of (1.1)×(0.5)×(0.2) meter^3 which is the entry for the trunk dimensions.

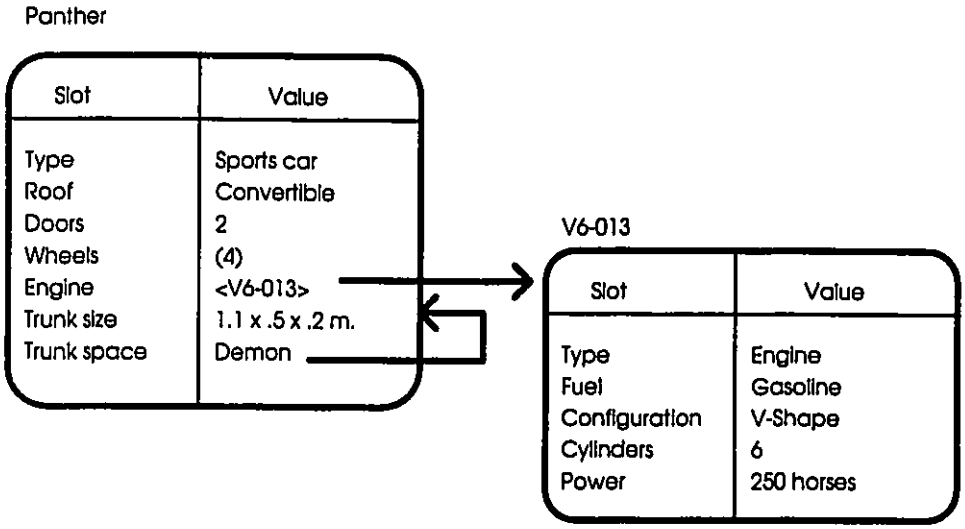


Figure 4.3 An example of a frame (Kim, 1991)

In prolog, the same frame can be expressed as follows:

```
:-op(100, xfx, '::').          % user defined frame operations: '::' and ':'
:-op(200, xfx, ':').

car :: [
        carName      :    panther,
        type         :    'Sports Car',
        doors        :    2,
        wheels       :    4,
        engine       :    v6_013,              % referring to another frame.
        trunkSize    :    [1.1, 0.5, 0.2],
        trunkSpace   :    trunkSpaceCalc(trunkSize)  % demon procedure that calculate trunk
                                                     % space with the input given as trunk size.
                          ].

v6_013 :: [
        type          : engine,
        fuel          : gasoline,
        configuration : v-shape,
        cylinders     : 6,
        power         : 250
                          ].

trunkSizeCalc(trunSize):-true.          % demon procedure.
```

Semantic Networks

One of the reasons for moving away from clausal form of logic as a representation scheme is the need to provide adequate facilities for defining complex constructs such as taxonomic relationships (De, 1988).

A semantic network is a graph whose nodes represent individual objects and whose directed arcs represent binary relationships. By using the following facts, Figure 4.4 illustrates an example of semantic net.

Facts :

Machine M_1 is idle at time t.
Machine M_2 is idle at time t.
Machine M_1 can perform operation o_1.
Machine M_1 can perform operation o_2.
Machine M_2 can perform operation o_2.
Machine M_2 can perform operation o_3.
o_1 is the first operation that must be performed on part P_1.

o_2 is the first operation that must be performed on part P_1.
o_3 is the first operation that must be performed on part P_1.

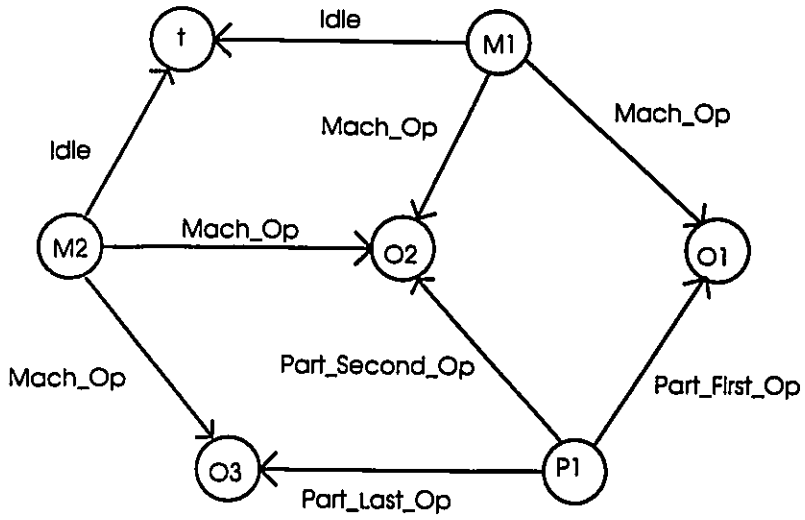
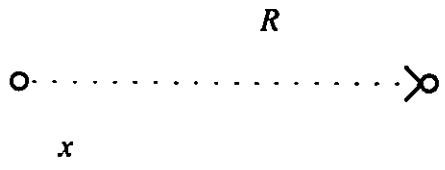


Figure 4.4 An example of a semantic network (De, 1988)

The semantic network representation of knowledge can be viewed as a graphical representation of the binary predicate version of predicate calculus, where an arc labeled *R* directed from node *x* to *y*



represents the assertion

$$R(x, y) \leftarrow$$

Hence the semantic network can be interpreted as the following assertions using first order logic rules:

ASSERTIONS:

IDLE $(M_1,t) \leftarrow$
IDLE $(M_2,t) \leftarrow$
MACH_OP $(M_1,o_1) \leftarrow$
MACH_OP $(M_1,o_2) \leftarrow$
MACH_OP $(M_2,o_2) \leftarrow$
MACH_OP $(M_2,o_3) \leftarrow$
PART_FIRST_OP $(P_1,o_1) \leftarrow$
PART_SECOND_OP $(P_1,o_2) \leftarrow$
PART_LAST_OP $(P_1,o_3) \leftarrow$

The Prolog representation is:

```
idle(machine_1, t) :- true.                 % Machine M1 is idle at time t
idle(machine_2, t) :- true.                 % Machine M2 is idle at time t
mach_op(machine_1, operation_1) :- true.    % Machine M1 can perform operation O1
mach_op(machine_1, operation_2) :- true.    % Machine M1 can perform operation O2
mach_op(machine_2, operation_2) :- true.    % Machine M2 can perform operation O2
mach_op(machine_2, operation_3) :- true.    % Machine M2 can perform operation O3
part_first_op(part_1, operation_1) :- true. % O1 is the first operation that must be performed on part P1
part_first_op(part_1, operation_2) :- true. % O2 is the first operation that must be performed on part P1
part_first_op(part_1, operation_3) :- true. % O3 is the first operation that must be performed on part P1
```

In semantic networks, nodes can be constants, variables or terms constructed by using function symbols. Arcs can represent conditions as well as conclusions and can be grouped into classes; then the overall graph is referred to as extended semantic network. The restriction to binary relationships is not an important limitation because every n-ary relationship can be expressed as a conjunction of n+1 binary relationships. For example, the 4-ary assertion

$$\text{BEGIN\_OP } (M_1, o_2, P_3, t_4) \leftarrow$$

which asserts that machine M_1 begins operation o_2 on part P_3 on time t_4, can be reformulated as five binary relationships:

e is an act of BEGIN_OP ←
e has actor M_1 ←
e has action o_2 ←
e has target P_3 ←
e has time t_4 ←

where *e* is assumed to be an "event". The corresponding semantic network can be given as in Figure 4.5.

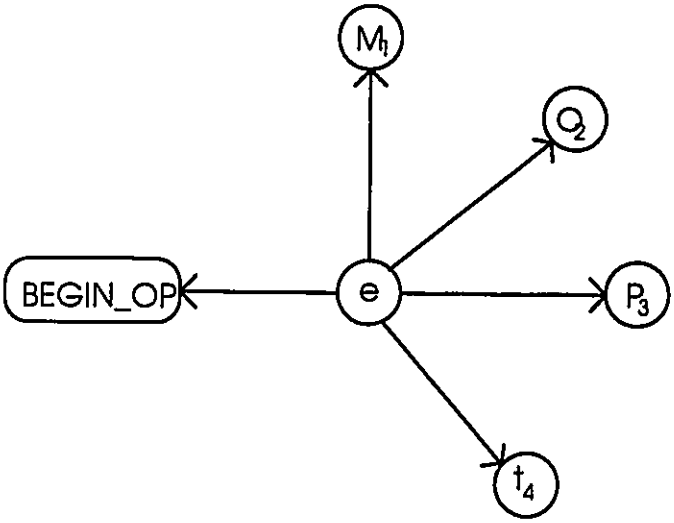


Figure 4.5 A binary representation of a 4-ary predicate (De, 1988)

The binary representation has some advantages. It is generally more expressive that the n-ary representation, and it makes it easier to add new information and ignore information that is unknown. When the components of a relationship is unknown, binary representation is also more convenient.

Object-Oriented Knowledge Representation

Structured representation languages provide a tool for building computational objects that exhibit the structure and behavior of their real-world counterparts. This is the property of homophormism: the representation is a direct image of the world it is intended to describe; it allows programmers to think about the formal representations in the

knowledge base in the same terms as they think about objects in the domain (Luger, 1989).

Multiple slots and values can be used to represent different aspects of a domain object. Values bound to slots describe the state of the object; attached procedures (demons) model both the behavior of domain objects and their interactions with each other. Procedural attachment adds considerable expressive power to these formalisms, allowing inherently procedural descriptions to be expressed in an appropriate language and integrated into the framework of a declarative representation.

In an object-oriented programming (OOP) language, everything is defined as an object or system of interacting objects. Individual objects are implemented as framelike structures whose slots contain either variables or procedural information. Procedures are expressed in an appropriate programming language, such as LISP, PROLOG, MODSIM II , or even a rule-based expert system language. Relationships between objects, such as causality, and the overall problem specification (e.g., what caused...?) are implemented as messages between objects. In addition, objects are organized into hierarchy of classes, allowing the inheritance of properties and procedures.

The basis of OOP is the ability to define computational objects of arbitrarily complex internal structure, which may be thought of subsequently as a single entity. These objects are not passive data structures (such as frames); they encapsulate both state and behavior. Their behaviors are implemented as attached procedures, called methods. These are invoked through messages sent to the object by the user or by other objects. On receiving a message, an object executes the designated method and responds accordingly. Thus, objects have characteristics of both data and programs in that they retain state variables as well as react procedurally in response to appropriate messages. They are active in the

sense that the methods are bound to the object itself, rather than existing as separate procedures for the manipulation of a data structure. Although frames allow the creation of complex objects and integration of procedural and declarative representations, they are passive data structures that must be acted on by external procedures. The execution of attached procedures requires that the procedure definition be retrieved and evaluated by some external agents. In an object-oriented language, objects execute their methods directly in response to a received message. Therefore, OOP provides an alternative to the traditional separation of data and procedure.

The other major feature of OOP languages is the support of classes and inheritance. In a pure object-oriented system, everything is an object; all objects belong to classes. This allows inheritance of slot names, values, and methods. In addition, each class object defines instance variables, which must be instantiated when an individual member of that class is created. Instance objects bind these variables to all the particular information, such as size, location, that distinguishes individuals from each other. The behavior of the members of the class, or the set of all messages to which the class responds, is called the protocol of the class.

In the design of knowledge bases, OOP provides a powerful and natural way to organize large and complex implementations. Besides the benefits of class inheritance for representing taxonomic knowledge, the message-passing aspect of object-oriented systems simplifies the representation of interacting components. In addition, object-oriented approach allows the knowledge engineers to deal with the complexity of the simulation. It enables them to build the model a piece at a time, focusing only the behaviors of the simple classes of objects. The full complexity of the system behavior emerges when the model is executed.

5 SIMULATION-BASED PRODUCTION SCHEDULING SYSTEMS

In the past, simulation has been used extensively for purposes of production system design and analysis with great success. In recent years, the possibility of using simulation in production scheduling has been recognized. Before giving the fundamental relationship between simulation and production scheduling systems, I would like to introduce basic concepts of simulation systems.

5.1 Basic Concepts of Simulation Systems

Simulation is experimentation with dynamic models, i.e., models which have time-varying behavior. It involves the generation of a behavior of a model of a system under some experimental conditions. This model usually takes the form of a set of assumptions concerning the operation of the system. These assumptions are expressed in mathematical, logical, and symbolic relationships between the entities, or the objects of interest and system. It provides insight into the sensitivity of the system to parameter changes, structure changes and input changes.

The needs and the advantages of simulation systems are: (1) once a simulation model is built, it can be used repeatedly to analyze proposed designs and policies, (2) a proposed system prior to implementation can be analyzed even the input data are somewhat sketchy, (3) experimenting on simulation models are much less costly than experimenting on real systems, (4) simulation methods are easier to apply rather than analytic methods. Whereas analytic models usually require many simplifying assumptions to make them mathematically tractable, simulation models have no such restrictions, and (5) in some cases, simulation is the only way of deriving a solution to a problem.

In general, there are two kinds of simulations based upon the characteristic of the events: (1) discrete event, and (2) continuous event. An event is defined as an instantaneous occurrence that may change the state of the system. The state of the system is defined to be that collection of variables necessary to describe the system at any time, regarding the objectives of the simulation experiment. The system, itself, is defined as a group of objects that are joined together in some regular interaction or interdependence to accomplish of the objectives. An entity is an object of interest in the system, an attribute is a property of an entity, an activity is a time period of specified length.

A discrete event simulation is the one in which the state variables change only at a set of countable number of points in time. Typical application areas are manufacturing, computer, military, inventory, and communication systems (CACI, 1984). In literature, usually a bank example is given to illustrate the idea. The state variable of the bank, say number of customers in the bank, changes only when a customer arrives or when the service provided a customer is completed in a point of time.

Discrete-event simulation is an event-oriented approach, and it is a classical way to describe individual routines (MODSIM II, 1991). In this case, no time passes during any event routine. Passage of time is handled by scheduling the next event routine for the object currently being manipulated. For smaller models, it is adequate, but in larger models it is often difficult to follow or modify the flow of logic which describes the behavior of an object. For that reason, process-oriented approach has been introduced. It simplifies larger models by allowing all of the behavior of an object in a model. Statements of a process can be expressed sequentially in a manner analogous to the system being described much more easily than event-oriented approach. Therefore it facilitates for the passage of time at one or more points in the method.

A continuous simulation is the one in which the state variables change continuously over time. Typical application areas for continuous simulation are control systems, chemical process representation, missile and aircraft simulation, power plant dynamics, biomedical systems, microprocessor controllers, fluid flow, and so forth (ACSL, 1991).

A simulation system can be thought as a software which is used in the analysis of the proposed systems. As a software, the simulation system must have some certain features to perform simulation task. Haider and Banks (1986), and Banks, et al. (1991) discuss the important features of a simulation system. Look at the Figure 5.1. In general, five basic features may be count: (1) input features, (2) processing features, (3) output features, (4) environment features, and (5) cost effectiveness.

One of the most important input features of a simulation system is an interface to other languages when the analyst desire to model the same problem with more details than with the simulation system he can. Inputs to the simulation system must be done interactively and/or in batch mode with an user interface to guide the user not to make a lot of input mistakes. The syntax of the modeling terminology must be easily understood by the user and should be consistent not to cause ambiguous.

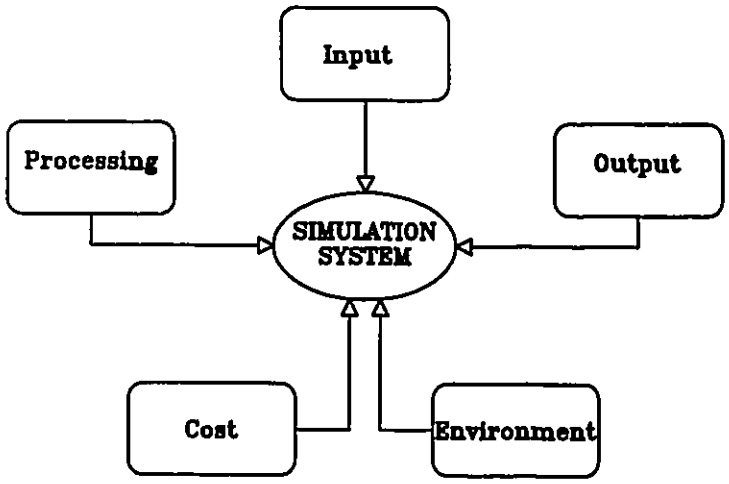


Figure 5.1. Five important features of a simulation system

For the processing features of a simulation system, it must have random variate generators such as exponential, uniform, and normal distributions. Analyst must be able to improve the accuracy of the representation of the system under investigation by using system objects (e.g., robots, cranes, conveyors, etc.). Global and local variables must be available for all transactions occurring through the system. Execution speed of the system must be reasonable to run many scenarios.

As output generation, the simulation system must produce customized reports as well as standard reports. Computation and display of some specific performance results could be created by tailoring the outputs to produce managerial level reports. Output of the system, with creation of some files, could be input for other software applications such as database.

The system must be cost effective, not very expensive than the gain comes from the use of simulation. Adequate hardware and software selection must be done properly to minimize the cost of the system. In order to reduce the training cost of the use of the

system, environment features must be developed as well. Documentation, ease of use, ease of learning, animation capability, on-line help are the most important environment features of a simulation system.

Next, I am going to discuss the relationship between simulation and production scheduling systems.

5.2 Simulation and Production Scheduling

In the last decade, there is an increase of the number of the papers issued about solving production scheduling problems with the help of simulation techniques. In literature, a system that solves production scheduling problem with the simulation techniques is called *Simulation-based Scheduling System*. In the next section, a number of simulation-based scheduling systems will be discussed from the literature. In this section, I will discuss about the reason for the emergence of the simulation-based scheduling systems. In addition, some basic concepts of simulation-based scheduling systems will be given.

5.2.1 Reasons for Emergence of Simulation-based Production Scheduling Systems

There are several reasons for the emergence of simulation-based scheduling systems. One may count the reasons as follows:

(1) Conventional production scheduling models are not sufficient to represent a realistic factory model. They usually underestimate the complexity of the manufacturing environment constraints with many simplifying assumptions to make them mathematically tractable, simulation models have no such restrictions.

(2) In fact, there are no certain way to represent the relationships between the objects in the factory environment, therefore a realistic model cannot be formulated mathematically. For that reason, realistic scheduling problems cannot be solved algorithmic methods with the use of traditional programming techniques.

(3) In addition conventional production scheduling techniques cannot manipulate unexpected events in a dynamic manufacturing environment such as machine break down, material shortage and so forth. If an unexpected event occurs, the available schedule will be invalidated. In order to obtain a new valid schedule, analyst need fast and reliable tool on which he will be dependent when he handles the changes occurred due to unexpected events.

(4) On the other hand, a simulation-based scheduling system can represent a realistic factory environment, and can handle unexpected events generating alternative schedules. Once a simulation model is built for a scheduling problem, it can be used repeatedly to analyze and verify proposed designs and alternatives even the input data are somewhat sketchy.

5.2.2 Basic Concepts of Simulation-based Production Scheduling Systems

Wyman (1991) introduces basic concepts of simulation-based scheduling systems. He says that a broad range of companies in the world are recognizing simulation-based scheduling systems as both pragmatic and effective finite scheduling systems. He emphasizes that the word *finite* stems from the recognition that resources and capacity in a factory environment are limited, as opposed to material requirements planning-based scheduling systems.

Material requirements planning (MRP) schedules are not verified for the feasibility within the system. It is assumed that sufficient capacity and other resources are available to carry out the schedule. MRP systems gradually evolved into the more comprehensive MRP-II systems (Manufacturing Resource Planning) precisely in response to the shortcoming. MRP-II systems has a feature for ensuring the feasibility of the MRP-based schedule with respect to available production capacity, called Capacity Requirements Planning (CRP) which deals with the resources as they are finite. Although CRP aims to verify the MRP-based schedule, its decision data is generated based on rough cut capacity estimations with over-simplifications, and with ignores of a realistic work-in-process inventories.

In simulation-based scheduling systems, more realistic system modeling can be done. The basic concepts simulation-based scheduling are given by Wyman (1991) by means of finite capacity scheduling such as: (1) capacity of work centers, (2) valid calendars, (3) jobs and tasks, (4) standard routings, (5) job specific detail, (6) system constants, and (7) scheduling phases.

Capacity of the work centers are limited as the resources constrain the production. The resources may be machines, human operators, tools, buffers or material handling equipment. Calendar of the plant is also limited, and the plant has different level of capacities for different shifts. Holidays and weekends capacities must be taken into consideration accurately.

The job orders that must be performed in the factory floor may consist of several tasks or operations. And each job must have a master record with detail information about each operation to be performed. Standard routings guide the operations for standard work

centers, setup times, processing times, alternative work center information and resources required. In addition, job orders must be identified in terms of customer ID, order number, part number, due date, release date, quantity and so forth.

The scheduling system must have some system constants such as the schedule start date, standard number of time units per shift, processing time units, and other background constants (e.g., output report formats, etc.). And scheduling cycle time phase must be set for rescheduling (e.g., weekly, or biweekly, etc.).

5.3 An Overview of Simulation-based Production Scheduling Systems

MacFarland and Grant (1990) have prepared a tutorial about a simulation based production scheduler, called FACTOR. The FACTOR simulator is coded entirely in C programming language. It is developed to meet the needs of the scheduling environment. It provides a rich set of standard modeling components of production systems (e.g., order status, production calendar, resources, parts, materials, process plans, shift schedules, etc.), and therefore, the flexibility to model a given production system to any required level of detail.

In FACTOR, resources are user definable to model machines, material handlers, operators, fixtures, etc. Moreover, resources may create some groups among them according to the some certain similarities (e.g., task, setup time, processing time, shift, or other details). Process plans provide the information of how and in what order operations are to be performed on each part. It consists of job steps (e.g., move, setup, operation, assemble, produce, inspect, etc.), which may have resource requirements, and has a duration that will take place during the execution of the step, with provisions for standard and alternative routing at any point.

In real life FACTOR is used to schedule of a short term production interval (e.g., a shift, a day, a week, etc.), and to handle unexpected events as they occur. According to the loaded current factory status to the FACTOR database, one can simulate the scheduling and review the reports. Potential scheduling problems can be corrected and then these new alternatives are re-simulated until an acceptable schedule has been reached (i.e. *what if?* analysis).

A simulation-based production scheduling system of a facility in a process industry is given by Barnes and Gardner (1990). The scheduling logic for the simulation model is designed to meet the scheduling objectives while adhering to the production constraints of the facility. The objectives are to satisfy demand, to meet due dates for final products, and to increase throughput by reducing the number of setups. They describe a simulation-based scheduling system for finite capacity scheduling of a facility in a process industry. The system is built on the framework of the FACTOR base system. They emphasized that simulation-based finite capacity scheduling systems have been successfully implemented in the manufacture of discrete parts. In addition to that, the adaptability of simulation modeling to the production constraints and continuous flows of materials found in process industries can provide an acceptable production schedules.

Mebarki (1993) and his colleagues have proposed an approach to dynamically select the best dispatching rule according to the production objectives and to the shop state in a Flexible Manufacturing System (FMS). They have developed a prototype of simulation tool for a FMS, in C++ on PC, which models real-time scheduling. It has been implemented by using simulation tool features provided by SIM-PP, an object-oriented process-oriented simulation software developed in C++.

In FMS environments, dispatching rules are widely used in real-time. No rule is globally better than the other. The performance of the rule depends on the state of the job, production objectives and the configuration of the system. In order to select the best dispatching rule regarding to the state of the job, the state of the system must be diagnosed first to correct the behavior of the system according to the production objectives.

Mebarki and his colleagues' dynamic scheduling approach selects the best dispatching rule considering several criteria. These are the objectives to be achieved (e.g., minimize flow time, minimize tardiness, minimize work-in-process inventory cost), the configuration of the shop (e.g., number of machines, size of the buffers), the production to be carried out (e.g., routings, number of operations, operating times, due dates, etc.), and the state of the shop (e.g., too many late jobs, that job is urgent, etc.).

6 SIMULATION-BASED EXPERT PRODUCTION SCHEDULING SYSTEMS

In this section, I will introduce the concept of simulation-based expert production scheduling system. It is the simulation-based scheduling system which is combined with an expert system together to solve production scheduling problem. First, I will give basic concepts of such systems. Second, an overview of the simulation-based expert production scheduling systems will be given from the literature.

6.1 Basic Concepts of a Simulation-based Expert Production Scheduling System

The recent developments in computer aided problem solving techniques affected all areas in engineering such as manufacturing engineering. The use of artificial intelligence and simulation techniques in production scheduling problem is a new dilemma for the researchers. There are several reasons behind this trend: (1) traditional programming techniques can help only conventional production scheduling models, (2) Conventional production scheduling models are not sufficient to represent a realistic factory model. They usually underestimate the complexity of the manufacturing environment constraints with many simplifying assumptions to make them mathematically tractable, simulation models have no such restrictions. In fact, there are no certain way to represent the relationships between the objects in the factory environment, therefore a realistic model cannot be formulated mathematically. For that reason, realistic scheduling problems cannot be solved algorithmic methods with the use of traditional programming techniques, (3) conventional production scheduling techniques cannot manipulate unexpected events in a dynamic manufacturing environment such as machine break down, material shortage and so forth. If an unexpected event occurs, the available schedule will be invalidated. In order to obtain a new valid schedule, analyst need fast

and reliable tool on which he will be dependent when he handles the changes occurred due to unexpected events, and (4) some production scheduling problems do not respond to algorithmic solutions, but with the traditional programming techniques you cannot apply heuristic approaches. In general, conventional scheduling techniques are based on lots of assumptions to make the problem solvable in traditional ways which deals with the scheduling problems in an unrealistic manner.

In order to overcome these problems, it is suggested three production scheduling systems: (1) knowledge-based production scheduling systems (or expert systems), (2) simulation-based production scheduling systems, and (3) simulation-based production scheduling systems combined with expert systems, called simulation-based expert production scheduling system.

With the combination of simulation and knowledge-based scheduling systems, the scheduling system will have the characteristics of an expert system and a simulation system. The system can show learning behavior which can be possessed with generalization of a current problem, and storing the generalized problem in learning base for future use for, or updating the knowledge base in real time. Complex problems that do not respond to conventional algorithmic solutions can be solved with the help of heuristic approaches, with numerous options and factors. Because, knowledge-based production scheduling systems allow to use heuristics, and rule-of-thumbs which are the simplifications that help limit the search for possible problem solutions. Symbolic processing, object-oriented software engineering techniques can be applied. Instead of algorithms, symbols can be used to represent real world objects, and characteristics associated with them. Moreover, complex relations among the objects can be easily modeled. A simulation-based scheduler can represent a realistic factory environment, and can handle unexpected events generating alternative schedules. Once a simulation model

is built for a scheduling problem, it can be used repeatedly to analyze and verify proposed designs and alternatives even the input data are somewhat sketchy. Time events can also be represented as objects, therefore dynamic plant behavior can be easily simulated. The models can be continuously updated to keep them current.

6.2 An Overview of Simulation-based Expert Production Scheduling Systems

A simulation-based expert production scheduling system in a flexible machining environment is developed by Bruno et al. (1986). The scheduling system consists of two subsystems, namely, an expert system for schedule generation and a simulation system for schedule evaluation. Due to the different data structure requirements by different flexible manufacturing system modules, separate data structures are produced for the expert and simulation subsystems.

Arons (1990) has proposed two approaches in real-time expert scheduling in a metal cutting manufacturing plant, called Esprit 809, with the help of simulation. First, the expert scheduler consists of a rule-based expert system and a simulation program which are two separated programs, working closely together. The second approach is an object-oriented knowledge-based system which carries out both the expert scheduling and simulation task itself.

He tried to solve the problem of scheduling a sequence of n jobs over a number of m machines in an optimal manner with respect to some production objectives. In the first case, the expert system contains only simple and unscolar knowledge necessary to obtain a feasible schedule. In order to improve the schedule quality, a simulation program is externally used. There are also various programs allowing communication with the outside of the world and database management system. For the second case, he proposed

a model-oriented knowledge-based system to improve the efficiency of the expert system. Hence, based on the model orientation, he could formulate explicitly all kind of underlying relations among the entities in the manufacturing environment.

These two cases has been used to improve production control for small batch part manufacturing systems to achieve both a more flexible use of resources and a decrease of the size of economic batches. Under the production control system scheduling, production control and monitoring-diagnostics functions are integrated together as separate components (modules). The main components are Scheduler/Simulator, Expert System, Dispatching, Monitoring-Diagnostics, Workstation Control, and I/O Interfacing. The both components, Scheduler/Simulator and Expert System, together form the *expert scheduler*. In general the system works as follows:

When production orders come from system level, the production control system activates the Scheduler/Simulator. The Scheduler/Simulator produces a very detailed production schedule in which the sequences of jobs which have to be performed on the machines have been arranged. This output is evaluated in the component Monitoring-Diagnostics. The output of Monitoring-Diagnostics system is passed over the Expert System to handle any disturbance which will invalidate the current schedule. The resulting output is passed over the Scheduler/Simulator, and the necessary changes are done. The Dispatcher takes its input from the Scheduler/Simulator. It releases the corresponding jobs to the appropriate machines on scheduled times. Moreover, it can take care of supplying the necessary tools and information. The Workstation Control is fed by the Dispatcher's output. The concerning jobs are controlled by this module of the production control system. Any unexpected event is immediately passed over the Monitoring-Diagnostics system component to trigger Expert System to handle the disturbance. I/O Interface

component is used for the communication between the external storage systems (e.g., material-tool inventory, etc.).

Expert System has distinct tasks to support Scheduler/Simulator. It analyses events during the manufacturing to determine whether to reschedule, or not. It determines the priority rules for work cells and machines. It can decide the best schedule among the generated schedules to execute as a current schedule. It has all relevant shop floor information (e.g., which machines are present), part routings (e.g., job sequences of each part, job descriptions), information concerning the progress of the manufacturing processes (e.g., which jobs are currently active at any moment, which event took place at any moment), management level plans (e.g., shifts, planned breaks). Therefore it has the capability to react quickly on events which invalidate the current schedule.

Fargher and Smith (1992) described a planner subsystem, a scheduler subsystem and a simulator subsystem of an advanced Computer Integrated Manufacturing (CIM) environment. The CIM environment responsible for coordinating all parts of a microelectronics manufacturing plant. The system architecture is based on a distributed object-oriented framework made of several cooperating subsystems which are Process Control, Modular Processing (for controlling the processing equipment), Generic Equipment Model (an interface between processing equipment and the rest of the factory), Specification (for factory documents and product specifications), Simulator (for modeling the factory for analysis purposes), Scheduler (for scheduling work on the shop floor), and Planner (for planning and monitoring of orders within the factory).

The planner is responsible to plan and predict work completion dates, given a set of goals and the current state of the factory with a required confidence level. The resource allocation is continually updated over time to reflect unexpected events such as machine

failures. The planner is capable to warn the user of the impact of the unexpected events (i.e., work completion dates are shifting). It can also perform "what-if" analysis by changing plans (i.e., user may want to hold or introduce a new machine). Finally, the planner can constraint work release into the factory based on the current plan being executed. Excess work release leads to increase in WIP, and thus early completion of work is undesirable. The scheduler is responsible for work release by a request coming from the planner. The planner influences the schedule being executed by constraining work release and predicting work completion times, which may be used in scheduler's dispatching rules. Another role of the scheduler is to make sequencing decisions for work on the factory floor (e.g., queue sizes, machine setups, etc.).

The simulator performs simulation using factory conditions possibly selected at random from a set of work release or machine failure distribution. The goal of the simulation may include the effect of introducing new machines into the factory, or re-training of the several operators. But the real important goal is accomplished by interacting with the planner. The planner can provide a work release plan which may be generated using initial factory status. This provides the simulator a work release time table. The simulator can compare the predicted completion times of the jobs with the results obtained by itself. This is important for verifying the plan of the system.

7 SIMEX: A SIMULATION-BASED EXPERT PRODUCTION SCHEDULING SYSTEM

In this chapter, I will describe a simulation-based expert production scheduling system, called SIMEX. First, the problem definition and the requirements of the problem for the scheduling system will be given. Second, the architecture of the SIMEX scheduling system will be introduced.

7.1 Problem Definition and Requirements

Our objective is to find a feasible schedule for a given production period that will correspond less holding cost of the material that will be used in the incoming production period and higher machine utilization in a dynamic manufacturing environment.

Before operating conditions and constraints of a dynamic plant is presented, the following notation and definitions are introduced:

E	denotes set of all unexpected events
I	denotes set of all operations
J	denotes set of all machines
K	denotes set of all parts
isn_j	denotes input storage number for machine $j, j \in J$
osn_j	denotes output storage number for machine $j, j \in J$
iss_j	denotes input storage size for machine $j, j \in J$
oss_j	denotes output storage size for machine $j, j \in J$
z_i	denotes material to be used in operation $i, i \in I$
w_i	denotes material weight to be used in operation $i, i \in I$

p_i	denotes unit price of material used in operation i, $i \in I$
s_i	denotes setup time for operation i, $i \in I$
s_j	denotes setup time for machine j, $j \in J$
P_k	denotes product number (or name) k, $k \in K$
DP_k	denotes description of the product P_k
SL_k	denotes stock level of product P_k
IP_k	denotes set of operations belonging to product P_k, $k \in K$
$\vec{IP_k}$	denotes a vector of triplets of operations belonging to product P_k, $k \in K$
$N\vec{IP_i}$	denotes number of operations in vector $\vec{IP_k}$
$\vec{DO}$	denotes a vector of daily orders
O_{ndx}	denotes identification number of an order
Q_{Ondx}	denotes quantity of an order O_{ndx}
DD_{Ondx}	denotes due date of an order O_{ndx}
OAD_{Ondx}	denotes order activation date of an order O_{ndx} (or manufacturing start time decided by SIMEX)
CD_{Ondx}	denotes current date when order O_{ndx} is entered to SIMEX
$\vec{PP}$	denotes a vector of a process plan
PP_{ndx}	denotes process plan index
DPP_{ndx}	denotes description of the process plan $\vec{PP}(PP_{ndx}, -, -, -, -)$
$\vec{XE}$	denotes a vector of unexpected event e, $e \in E$

The operating conditions and constraints of a plant may be described as follows:

(1) There are *daily orders*, $\vec{DO}$. Contents of an order are depicted in Table 7.1. A daily order $\vec{DO}$ of a product P_k is a vector containing order index, O_{ndx}, product number,

P_k, order quantity, Q_{Ondx}, due date, DD_{Ondx}, order activation date, OAD_{Ondx}, and current date CD_{Ondx}, e.g.,

$$\vec{DO} = (O_{ndx}, P_k, Q_{Ondx}, DD_{Ondx}, OAD_{Ondx}, CD_{Ondx}) \tag{7.1}$$

Table 7.1 Contents of an order

An Order	Order Index
	Product Number
	Order Quantity
	Due Date
	Order Activation Date
	Current Date

The order index simply shows the order number. The order must be completed before the due date, which is shipping time to the customer. Order activation date, which is suggested by SIMEX, is the starting time of manufacturing of the ordered product. Product number is the code of the product.

(2) Each *product*, $\vec{P}$, is a vector containing description, DP_k, product number, P_k, stock level, SL_k, and process plan, PP_{ndx}. e.g.,

$$\vec{P} = (DP_k, P_k, SL_k, PP_{ndx}) \tag{7.2}$$

The contents of a product are indicated in Table 7.2.

Table 7.2 Contents of a product

Product	Description
	Product Number
	Stock Level
	Process Plan

The manufactured products must be used as soon as possible to reduce space requirements and to reduce product inventory holding cost (i.e., work-in-process cost). The raw material holding cost must also be minimum for the products that will be manufactured in a given production period.

(3) Contents of a process plan and an operation is indicated in Table 7.3.

Table 7.3 Contents of a process plan and an operation

Process Plan	Process Plan Index
	Process Plan Description
	Product Number
	No of Operations
	Operations
Operations	Operation Description
	Operation No
	Machine No
	Input Storage No
	Output Storage No
	Operation Setup Time
	Machine Setup Time
	Operating Time
	Input Storage Size
	Output Storage Size
	Material Name
	Unit Material Weight
	Unit Price of Material
	Next Operation

A *process plan* consists of the necessary information to produce the concerned product order. In other words, a process plan $\vec{PP}$, is a vector containing process plan

index PP_{ndx}, description of the process plan DPP_{ndx}, product number P_k, number of operations in the process plan $N\vec{IP}_k$, set of operations, IP_k, belonging to the process plan. e.g.,

$$\vec{PP} = (PP_{ndx}, DPP_{ndx}, P_k, N\vec{IP}_k, IP_k) \tag{7.3}$$

Having a process plan index enables to hold different alternative process plan to the product k. The number of operations enables to SIMEX, how many operation blocks are needed to take into account, and set of operations consists of the operations to be performed to achieve the product k. Description of the process let SIMEX know that process plan is the basic one or an alternative one. A basic process plan is a sequence of operations of a product that to achieve it requires the least time.

Each operation is an information block or a vector of triplets, each containing operation description D_i, operation number i, machine number j, input storage number isn_j, output storage number osn_j, operation setup time s_i, machine setup time s_j, operating time t_i, input storage size iss_j, output storage size oss_j, material name z_i, unit material weight to be used w_i, unit price of material p_{zi}. e.g.,

$$\vec{IP}_k = \{(D_i, i, j, isn_j, osn_j, s_i, s_j, t_i, iss_j, oss_j, z_i, w_i, p_{zi}), \ldots (D_{\tilde{n}}, \tilde{n}, j, isn_j, osn_j, s_{\tilde{n}}, s_j, t_{\tilde{n}}, iss_j, oss_j, z_{\tilde{n}}, w_{\tilde{n}}, p_{z\tilde{n}})\} \tag{7.4}$$

where $\tilde{n}$ is number of operations in IP_k (or $N\vec{IP}_k$).

(4) There are *j machines* in the plant to process the jobs. The machine requirements of each job are known. The machine setup times are needed for the preparation of machines for new jobs. Some machines are grouped. The basic idea of grouping is

to decompose the manufacturing system into subsystems (i.e. similar machines may create one group considering similarity of input materials. Therefore, some processes can be done only on one machine and some have a choice of up to the number of similar machines).

(5) The *production period* is d days long. Before starting to the production, a certain amount of time, called schedule generation and verification time, is needed to prepare a master production schedule to apply to the production period.

(6) A *master production schedule*, for a production period and for a set of job orders, consists of a detailed process plan which shows resource allocation as well as machine schedules.

(7) The plant experiences *fluctuations* arising from several causes such as: machine failures, products requiring rework, and changes in the job requirements. Each unexpected event is a vector containing event description ED_e, event number e, $e <$ 0, machine number j, input storage number isn_j, output storage number osn_j, operation setup time s_e, machine setup time s_j, operating time t_e, event start time t_e, event finishing time ft_e, input storage size iss_j, output storage size oss_j, material name z_e, unit material weight to be used w_e, unit price of material p_e. e.g.,

$$\vec{XE} = (ED_e, e, j, isn_j, osn_j, s_e, s_j, t_e, st_e, ft_e, iss_j, oss_j, z_e, w_e, p_e) \qquad (7.5)$$

These unexpected events quickly invalidate schedules, requiring reactive management to be made. The large amount of parameters, complex constraints and the frequent fluctuations make conventional approaches infeasible. Therefore, in order to meet the job requirements, the simulation-based expert production

scheduling system will perform mainly machine scheduling in response to updates in current plant status based on recognition of how a given status update affects the current schedule (e.g. what feasibility constraints are now violated, what opportunities for schedule improvements are now apparent), and what reactive action should be taken. The model must be quite flexible to handle any change in the operating conditions such as the machine capacities change when some machines are not in working conditions.

We have to prepare a machine schedule to correspond to the minimum holding cost of the material that will be used in the production period. We would like to achieve the production on time (i.e., finishing the production on or before due dates). If this is not possible, then we have to revise the job orders to get a feasible amount to manufacture. If the request for production scheduling is not feasible, the system would advise the user.

7.2 Solution Approach

SIMEX scheduling software is developed to solve real-time complex production scheduling problems. SIMEX is a simulation-based scheduling system combined with an expert system. The operating system for the software is assumed to have interprocess communication facilities (IPC) such as UNIX System V Release 4. The general framework of SIMEX is designed in this thesis as SIMEX runs on an operating system that has asynchronicity (e.g., one or more programs may execute independently of the other at the same time). SIMEX is an industrial production scheduler in real time, and asynchronous running is crucial for immediate action to handle occurring unexpected events (i.e., order cancellation, machine breakdown, etc.).

SIMEX, the prototype, is developed by using Microsoft Visual Basic, Microsoft Visual C++, MODSIM II, and Prolog compilers (Figure 7.1). Back-end interface and front-end

interface are developed by Visual Basic. Expert system module is coded by Prolog, and the remaining modules are written in MODSIM II, and Visual C++.

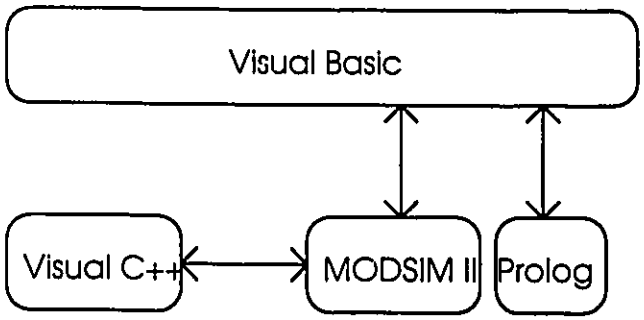


Figure 7.1 Languages used in the development of SIMEX

There are several reasons behind the use of different languages in a software. First, every language has different kinds of functionalities and programming strategies for different problems. Instead of reinventing the new functionalities for upgrading the programming language, it is less time consuming to use any other language that meets the requirements. Second, after having object code, it is easy to wrap up the programs by a graphical user interface.

In the case described above, Microsoft Visual C++ is externally used in MODSIM II for Windows. MODSIM II already uses Microsoft C compiler provided by Microsoft Visual C++ to generate C code. C language is capable to manipulate much faster the operating system objects such files, file related objects than MODSIM II does. Hence, Visual C++ is also used in the coding.

The choice of MODSIM II is due to its integrated structure of simulation, modular and object-oriented programming technologies which are gathered within a single language. Its simulation capabilities are provided in library modules. These modules provide direct support for all capabilities needed to program discrete event simulation models. Object-oriented programming features facilitates strong data encapsulation and abstraction in the creation of objects to be used in simulation events. Moreover, smart its smart compiler supports separate compilation of the different modules. This means that if you alter one module in a program, only that module needs to be recompiled. This is a powerful time-saving feature which greatly speeds up the program development and evolution.

The choice of Arity/Prolog, which is used for the development of the expert system module, is done because of its availability. The other reason is limited functionality of the expert system module of the prototype. Priority rules that construct the rule base are not sophisticated as much as a human expert. However, the sophistication of the rule base can be increased by adding more complex priority rules to the simulative scheduler heuristic algorithm. Those relatively more sophisticated rules can be coded in the rule base of the expert system by a knowledge engineer. This upgrade of rulebase can be done easily with the declerative knowledge representaion capability of the Prolog language.

For graphical user interface, Visual Basic is used because of its simplicity. During a development of any prototype, regularly occurrence of design changes is highly probable. In order to be fast and efficient, Visual Basic was the perfect choice. Moreover, it also helps for achieving a completed software for first release of the prototype. If the prototype is accepted, then it is the managerial decision for coding the user interface in any other language for the first product release.

For operating system environment choice of the prototype, MS Windows is one of the best available system that supports dynamic data exchange (DDE) and object linking and embedding (OLE). It is therefore much easier to develop a user interface that displays the detailed schedule results in graphics. Those functionalities ease to display data as well as dynamically updating the data. By those capabilities, SIMEX is quite flexible to be upgraded, and easy to be used as mentioned later.

The reasons behind the development of SIMEX can be counted as follows: (1) traditional programming techniques can help only conventional production scheduling models, (2) conventional production scheduling techniques fail to represent a realistic factory model, (3) unexpected events in a dynamic manufacturing environment cannot be manipulated with the conventional techniques, (4) availability of the recent tools for object oriented programming, simulation and knowledge-based systems development, and (5) availability of operating systems that enables interprocess communication.

7.3 Inputs/Outputs of SIMEX

In order to use the program, the system must be fed with some certain information of the manufacturing environment. In other words, we are tailoring the system to a plant. All detailed information about the factory must be entered to the system. The inputs are: (1) daily orders, (2) process plans, (3) inventory, and (4) shop floor information. They are all maintained in different databases with a front-end user interface access (Figure 7.2). In sections 7.1 and 7.4.1, detailed explanation of the input parameters are given.

The outputs are: (1) detailed schedules with performance measurements, and (2) auxiliary output. Detailed schedules are displayed with the help of back-end graphical user

interface to visualize the results to users. Auxiliary output is generated by the diagnostics module to inform the user about SIMEX's last action against unexpected events.

7.4 System Components of SIMEX

The system components of SIMEX are: (1) production environment control module, (2) simulative scheduler module, (3) expert system module, (4) verification module, and (5) diagnostics module. The components of the system is given in Figure 7.2. Each module is presented separately.

7.4.1 Production Environment Control Module

Production environment control module is a database to keep shop floor status (e.g., machines, tools, storage etc.) and inventory status (e.g., stock level, unit price etc.) information. The database can be updated daily with a user interface to keep it current. SIMEX has a user interface to get information related to factory status (Figure 7.3).

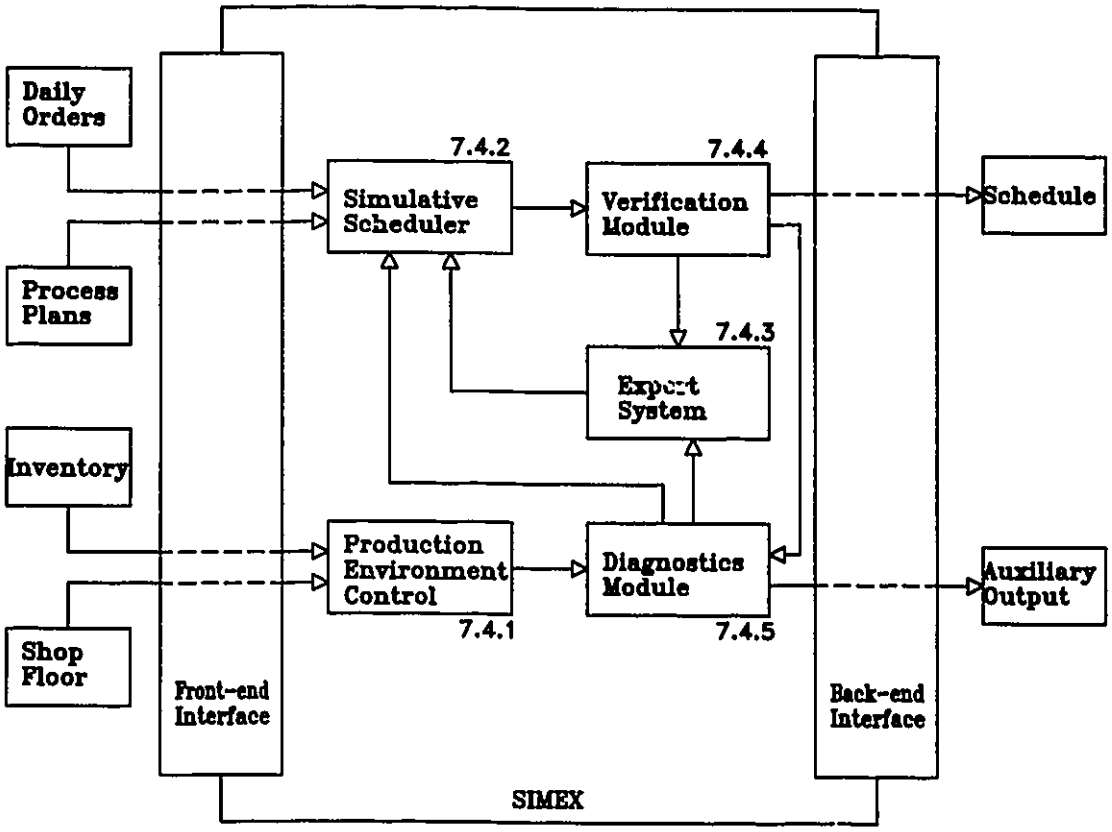


Figure 7.2 The components of SIMEX (a general framework)

In order to learn the machine utilization measurements of the first machine, just click M/C1 with the mouse. SIMEX will display two options: (1) detailed schedule or (2) machine utilization summary. And then select one of the displayed options. If you select machine utilization summary, then SIMEX will draft the following pie chart as a summary of utilization calculations (Figure 7.4).

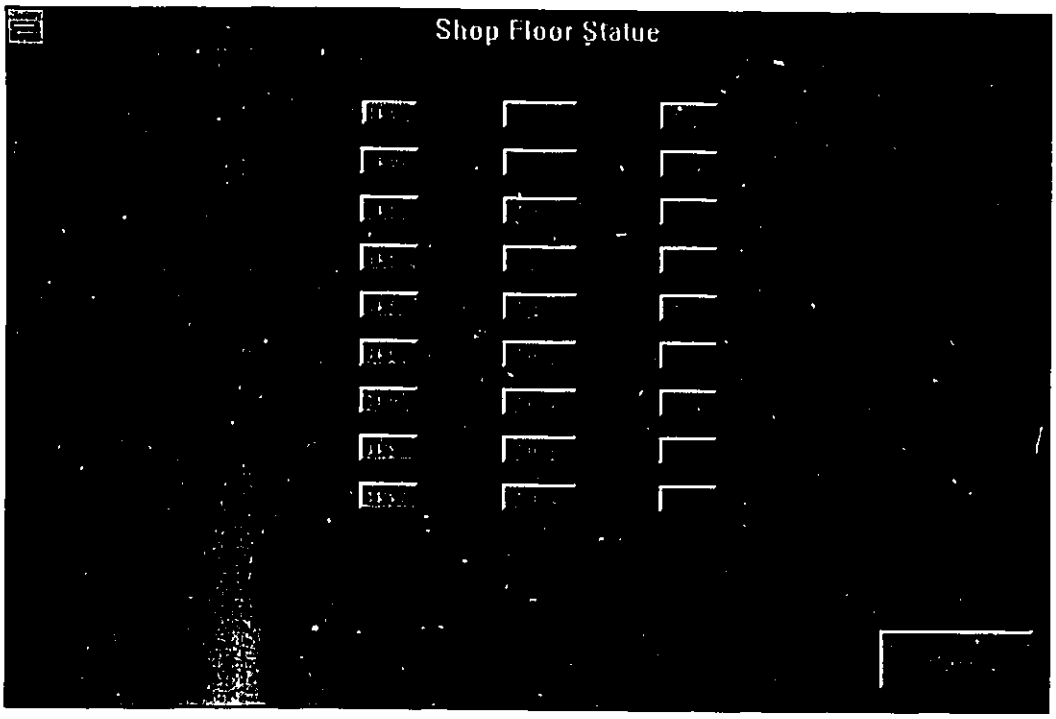


Figure 7.3 SIMEX's shop floor status screen

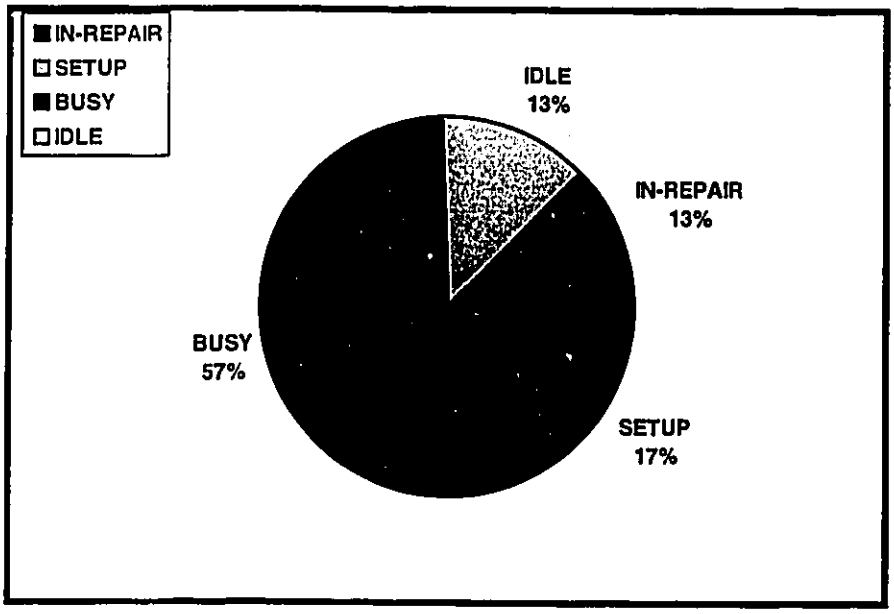


Figure 7.4 M/C1 machine utilization measurements

Process plans and orders are also included in the production environment database, and can be updated easily with a user interface. Figure 7.5 indicates the Order Maintenance Screen of SIMEX. Daily customer orders are entered to SIMEX via that screen. Order database can be accessed or updated easily with this screen and can be kept current. If user wants to cancel an order, s/he must use Unexpected Event Maintenance Screen to inform SIMEX (Figure 7.12). Because order deletion in Order Maintenance Screen just updates the order database, and ignored by SIMEX.

Figure 7.6 indicates the Process Plan Maintenance Screen of SIMEX. User easily can update the existing process plans or add new ones or delete the useless ones. With the *Search* button, you can reach immediately the required plan with the argument of process plan index.

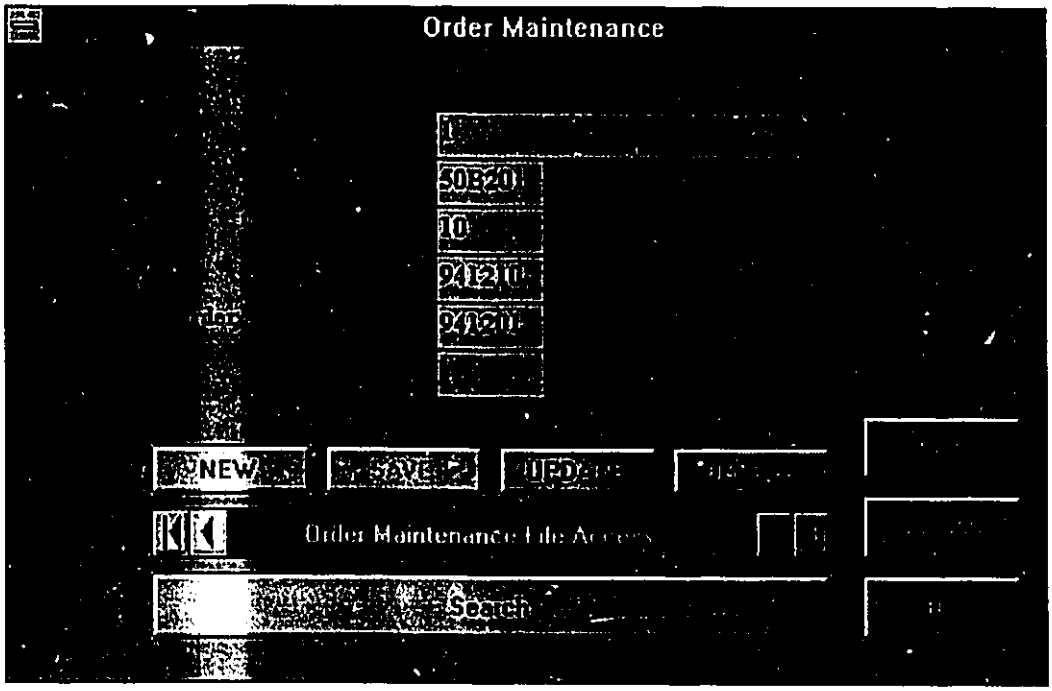


Figure 7.5 SIMEX Order Maintenance Screen

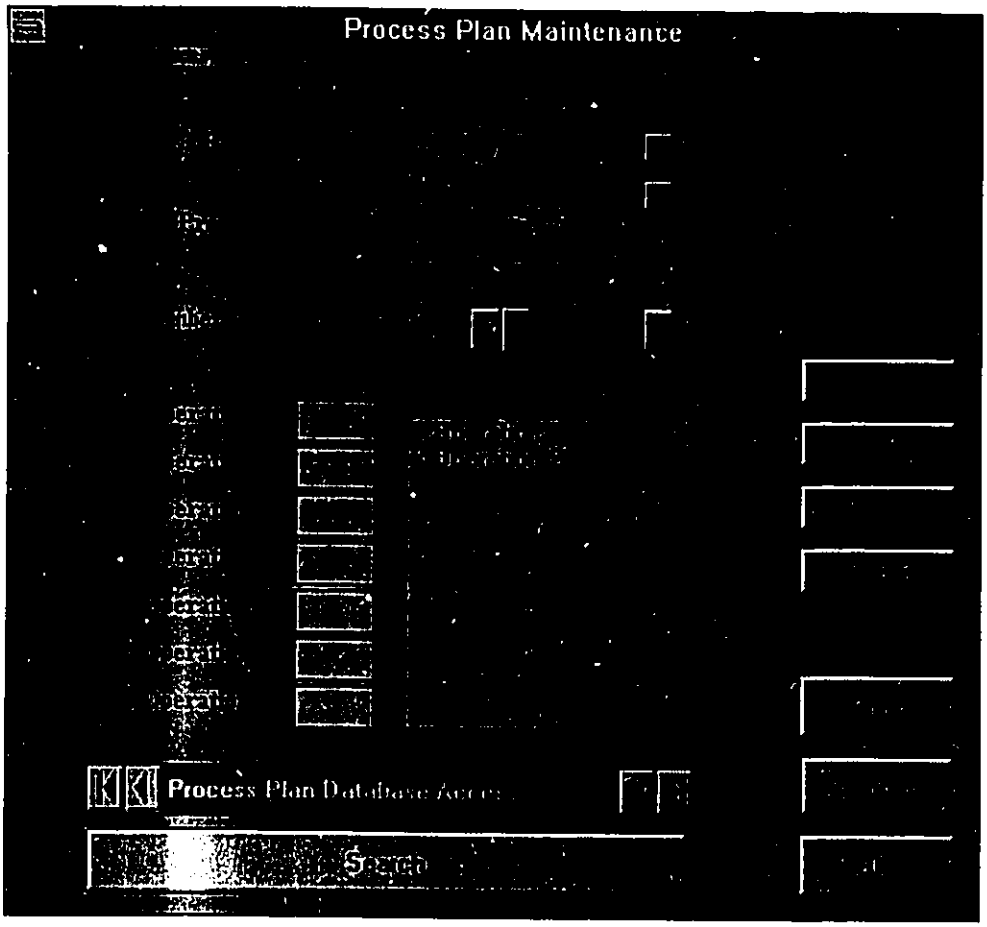


Figure 7.6 SIMEX Process Plan Maintenance Screen

The operation that entered to the process plan must exist in the operation database. Operation database maintenance screen is indicated in Figure 7.7. All screen must be filled. Again user can add, update, delete or search for a record easily.

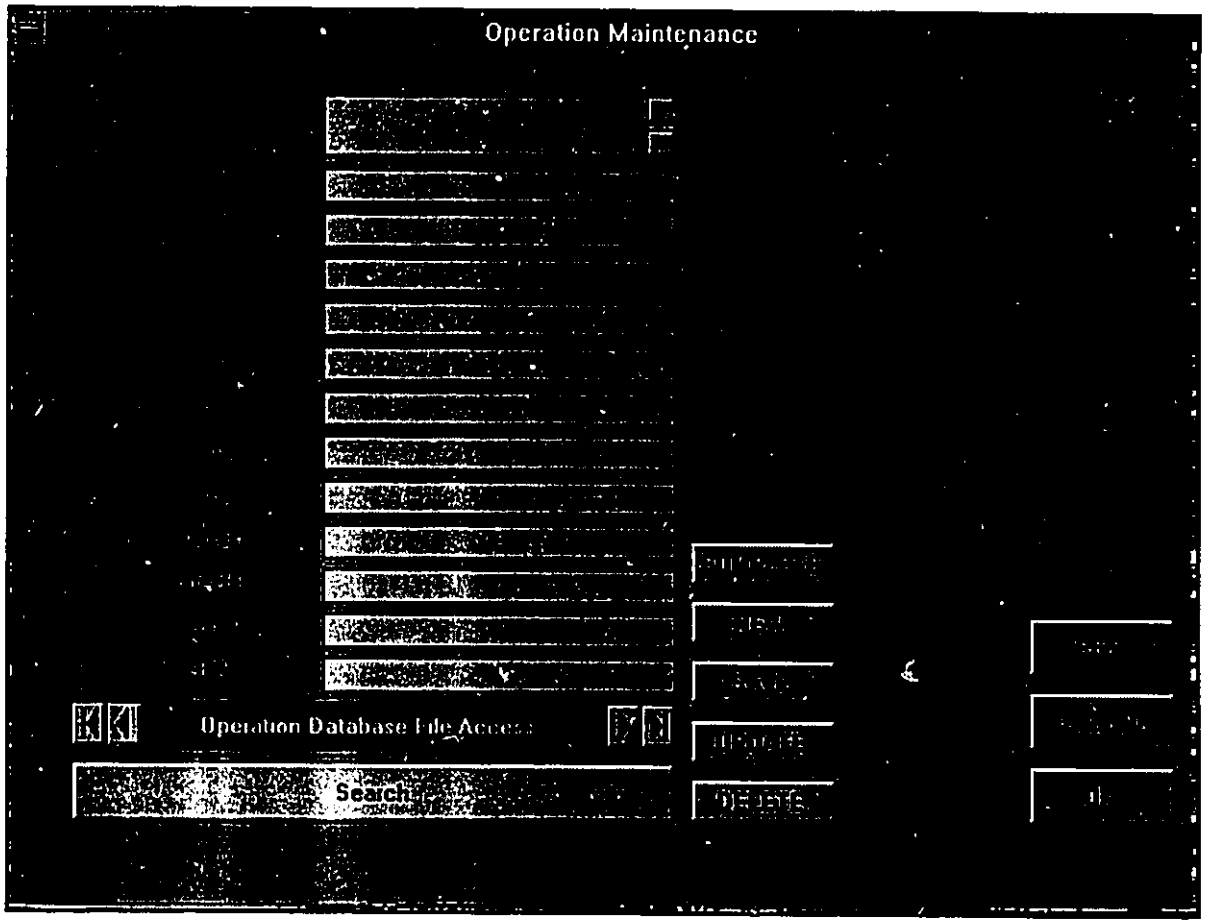


Figure 7.7 SIMEX Operation Maintenance Screen

Figure 7.8 shows product maintenance screen, and Figure 7.9 shows material maintenance screen. Again the required data can be accessed or updated easily by a user.

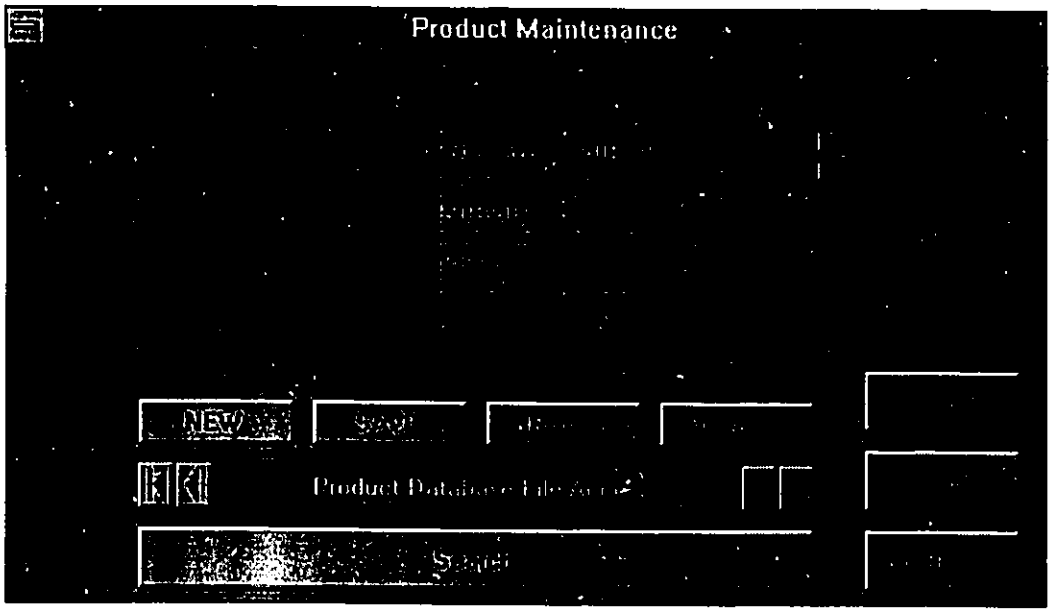


Figure 7.8 SIMEX Product Maintenance Screen

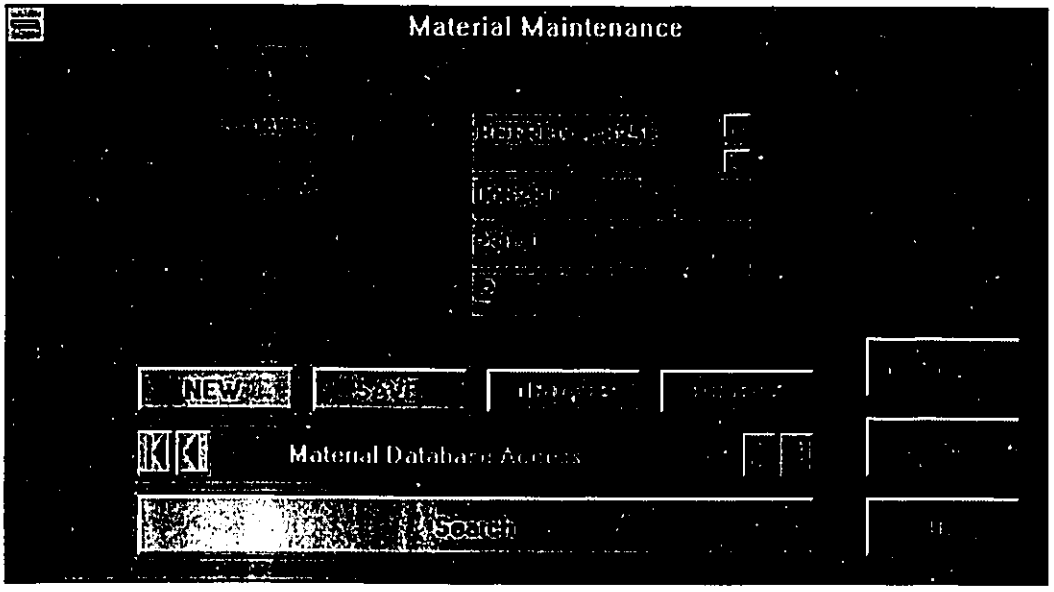


Figure 7.9 SIMEX Material Maintenance Screen

7.4.2 Simulative Scheduler Module

Simulative scheduler module performs the main production scheduling activity by simulation techniques and using a combination of priority rules and added heuristics. SIMEX starts its activities by activating the simulative scheduler module when the production orders come into the system. If the production orders are new orders, scheduling will be performed only those new orders without changing the current schedule. If some orders are canceled, rescheduling might be necessary to improve the current schedule. After scheduling is done, it passes the detailed job schedule to the verification module. The verification module detects unfeasible orders and calculates the performance measures of the objective criteria. If something cannot be accepted, the result will be passed over to the expert system to give the decision to reschedule or accept the current available schedule.

If the simulative scheduler module is activated for the first time, then it generates a schedule for the all product orders. During this process, it divides the scheduled orders into two categories: (1) Reschedulable, and (2) Non-reschedulable. A reschedulable scheduled order is the one that basic process plan is not used, and the difference between the completion time of an order and its due date is in the time interval [0, 1]. That means, if an order is completed within the last day of delivery with a basic process plan, then it is identified by SIMEX as a non-reschedulable order. In any product order cancellation, by rescheduling, the current schedule might be improved by using basic process plan. In other words, in any product order cancellation, the current schedule cannot be improved (including itself). The main reason behind this categorization is the desire to elimination of rescheduling time when a reschedule is necessary. The logical structure is depicted in the following Figure 7.10.

If First-Run	
then	• Schedule all orders • Divide the scheduled orders into two lists List 1: Reschedulable scheduled orders List 2: Non-reschedulable scheduled orders
else	• Reschedule reschedulable orders • Determine new nonreschedulable scheduled orders • Add new nonschedulable orders to the List 2 • Delete new nonschedulable orders from the List 1
End If	

Figure 7.10 Categorization of scheduled operations:
- Reschedulable
- Non-reschedulable

7.4.3 Expert System Module

Expert system module contains production rules (or priority rules) for the simulative scheduler algorithm. The rule base is accessible with an user interface. Some rules can be turned off according to the user request. The necessary changes automatically is coded in Prolog Language by SIMEX without any user interference (Figure 7.11).

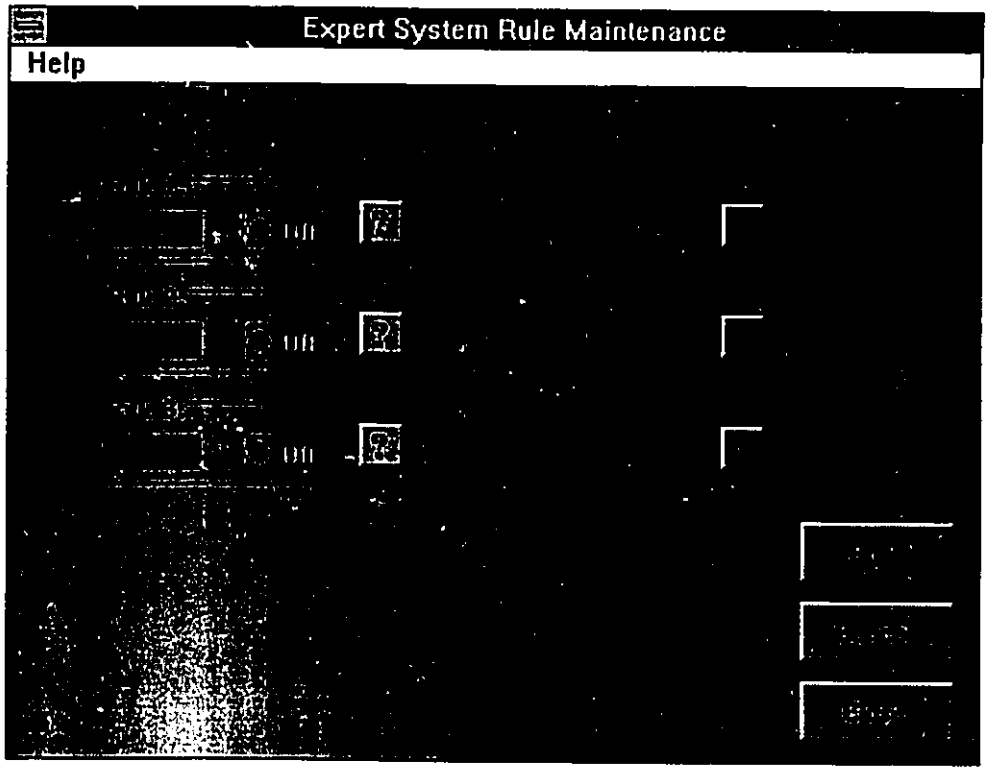


Figure 7.11 SIMEX Rule Base Maintenance Screen

Priority Rule Base Maintenance Screen gives an interactive change in rule base according to the users desire. User might desire to ignore one or more priority rules to obtain different and fast combinations of production schedules.

The priority rules can be given as follows:

Rule 0: Select least starting time among unexpected events.

Rule 1: Select an operation with an earliest due date.

Rule 2: Select an operation performed with a cheapest unit material cost

Rule 3: Select an operation such that same machine can operate on previous operation and the new operation, (e.g., eliminating the setups, selecting lowest required setup times).

Rule 4: Select an operation with a higher WIP level (e.g., don't start to create other product WIPs, before finishing one product type)

Rule 5: Select an operation with a largest number of successive operations.

Rule 6: Break tie, (e.g., arbitrarily choose one operation).

7.4.4 Verification Module

Verification module performs feasibility check of the schedule given by the simulative scheduler module. If there is any late product, it is monitored and displayed immediately. Moreover, manufacturing activation days of the orders are readjusted for possible improvements on the schedule by shifting the scheduled operation on the time horizon. If rescheduling is necessary, it informs the user.

This module also calculates the performance measures such as machine utilization, number of machine setups and material holding cost.

Resource utilization calculations are given as follows:

Busy percentage:

$$RU_{lq} = \frac{\sum_{i \in I} b_{ilq}}{\text{SIMTIME}}$$

for $\forall\ l, q$ where

RU_{lq} denotes resource utilization of qth resource type l, $l \in L$, $q \in Q$

b_{ilq} denotes busy time of operation i of qth resource type l, $l \in L$,

$q \in Q,\ i \in I$

Idle percentage:

$$RU_{lq} = \frac{\sum_{i \in I} IDLE_{ilq}}{\text{SIMTIME}}$$

for $\forall\ l, q$ where

$IDLE_{ilq}$ denotes idle time of operation i of qth resource type l, $l \in L$, $q \in Q,\ i \in I$

Failure percentage:

$$RU_{lq} = \frac{\sum_{i \in I} e_{ilq}}{\text{SIMTIME}}$$

for $\forall\ l, q$ where

e_{ilq} denotes unexpected event time of operation i of qth resource type l, $l \in L$, $q \in Q,\ i \in I$

Setup percentage:

$$RU_{lq} = \frac{\sum_{i \in I} s_{ilq}}{\text{SIMTIME}}$$

for $\forall\ l, q$ where

s_{ilq} denotes setup time of operation i of qth resource type l, $l \in L$, $q \in Q,\ i \in I$

Material Holding Cost:

$$\sum_{\psi}\left[(p_{\psi} \times w_{\psi}) \times \left(\max\{dd_{\psi}, \Phi_{\psi}\} - \{f_{\psi}\ div\ 86400\}\right)\right] \times (INT + 365)$$

where

p_Ψ	denotes price of material used in operation Ψ
w_Ψ	denotes weight of material used in operation Ψ
dd_Ψ	denotes due date of operation Ψ
Φ_Ψ	denotes actual delivery time of operation Ψ as a product
f_Ψ	denotes finishing time of operation Ψ in flow shop
INT	denotes interest rate

7.4.5 Diagnostics Module

The Diagnostic Module detects unexpected events whenever occurs, and result is passed over to the Expert System module to last decision. SIMEX activates *unexpected event impact* program to monitor the event influence to the current schedule. If the rescheduling is necessary, then the simulative scheduler will be activated (Figure 7.12).

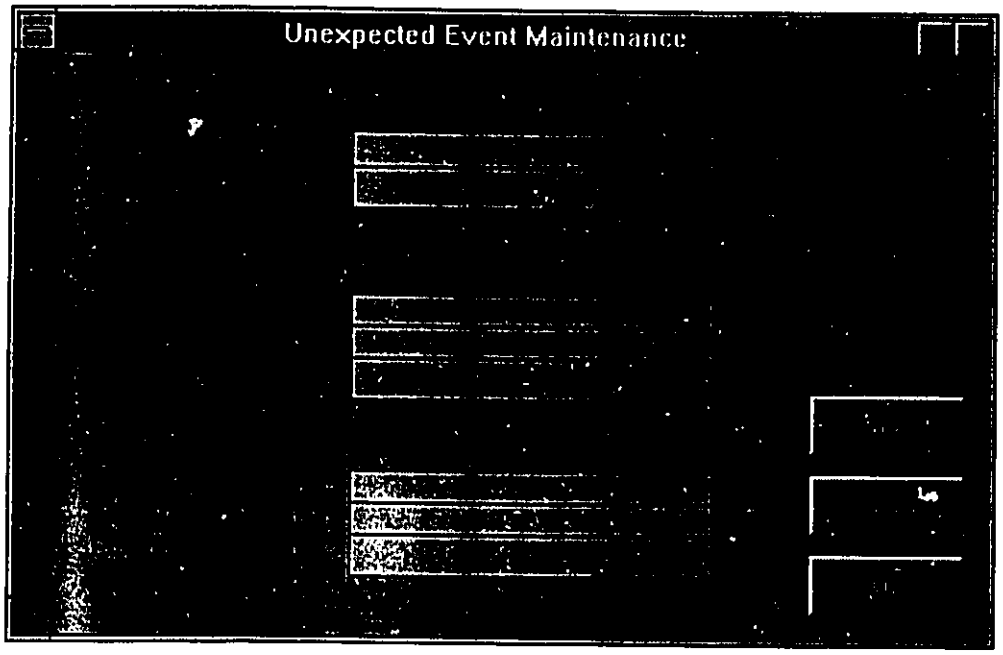


Figure 7.12 SIMEX Unexpected Event Maintenance Screen

There are mainly three kinds of unexpected event that SIMEX can handle: (1) Order cancellation, (2) material shortage, and (3) resource failure (i.e., machine failure, operator is seek, storage is full, etc). Whenever an unexpected event occurs, user must enter correct data to inform SIMEX by selecting correct option. Diagnostics module immediately responds to take necessary cautions.

7.5 Simulative Scheduler Heuristic Algorithm

Notation and definitions regarding to the simulative scheduling heuristic algorithm are as follows:

GI	denotes set of all generated operations by SIMEX
$[gi]$	denotes generated operation index,
l	denotes resource type,
	{1 : Machine}
	{2: Human}
	{3: Buffer-in}
	{4: Buffer-out}
L	denotes set of all resources, $L = \{1, 2, 3, 4\}$
q	denotes resource index,
$dd_{[gi]}$	denotes due date of generated operation $[gi]$, $[gi] \in GI$
$edd_{[gi]*}$	denotes generated operation that has earliest due date among schedulable generated operations
$bl_{[gi]}$	denotes current buffer level of generated operation $[gi]$, $[gi] \in GI$
$hbl_{[gi]*}$	denotes highest buffer level of generated operation $[gi]$ among schedulable generated operations

$rst_{[gi]}$ denotes required setup time for generated operation $[gi]$, $[gi] \in GI$

$lrst_{[gi]*}$ denotes generated operation $[gi]*$ that has lowest required setup time among schedulable generated operations

stx_e denotes least start time for an unexpected event $e \in Œ$

$lstx_e$ denotes an unexpected event that has least start time among others

$umc_{[gi]}$ denotes unit material cost of generated operation $[gi]$, $[gi] \in GI$

$cumc_{[gi]*}$ denotes cheapest unit material cost of generated operation $[gi]$ among schedulable generated operations

$nso_{[gi]}$ denotes number of successive operations of generated operation $[gi]$, $[gi] \in GI$

$lnso_{[gi]*}$ denotes largest number of successive operations of generated operation $[gi]$ among schedulable generated operations

$os_{[gi]}$ denotes status of generated operation $[gi]$, $[gi] \in GI$

{0: nonschedulable}

{1: schedulable}

{2: in-process}

{3: finished}

$rpt_{[gi]}$ denotes remaining processing time of generated operation $[gi]$, $[gi] \in GI$

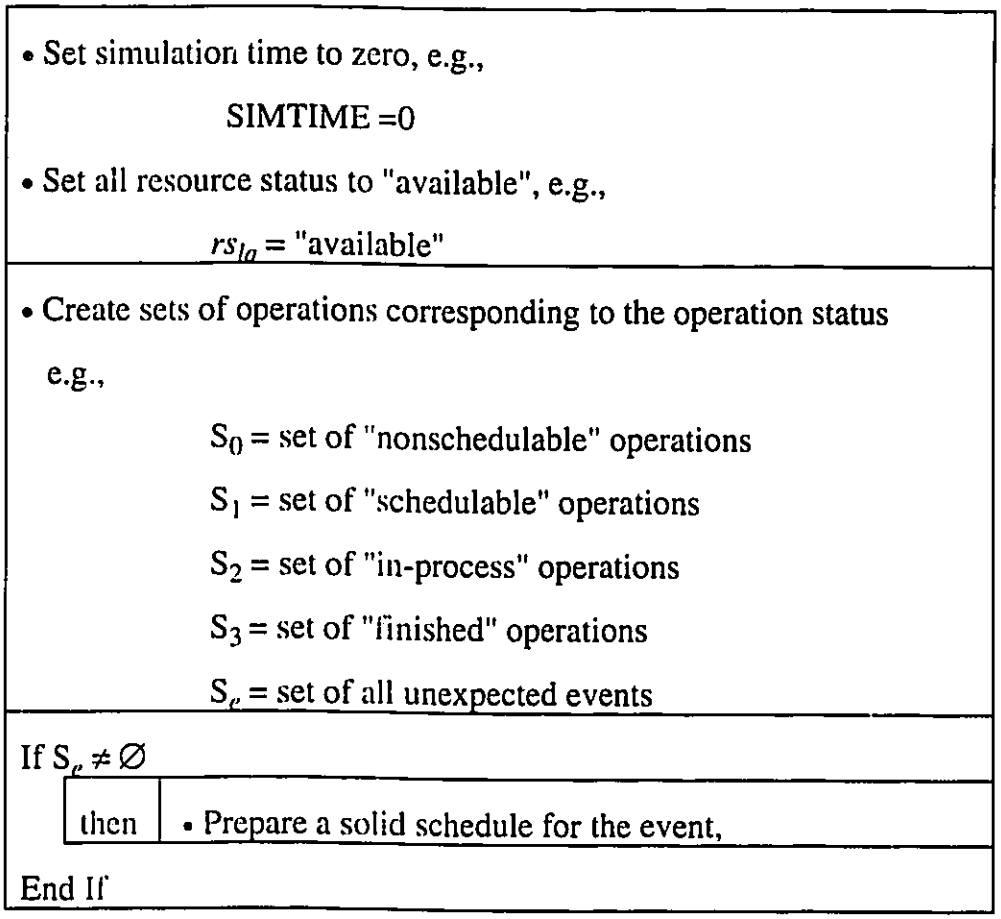
rs_{lq} denotes resource status of qth resource type l,

{-1: in-repair}

{1: available}

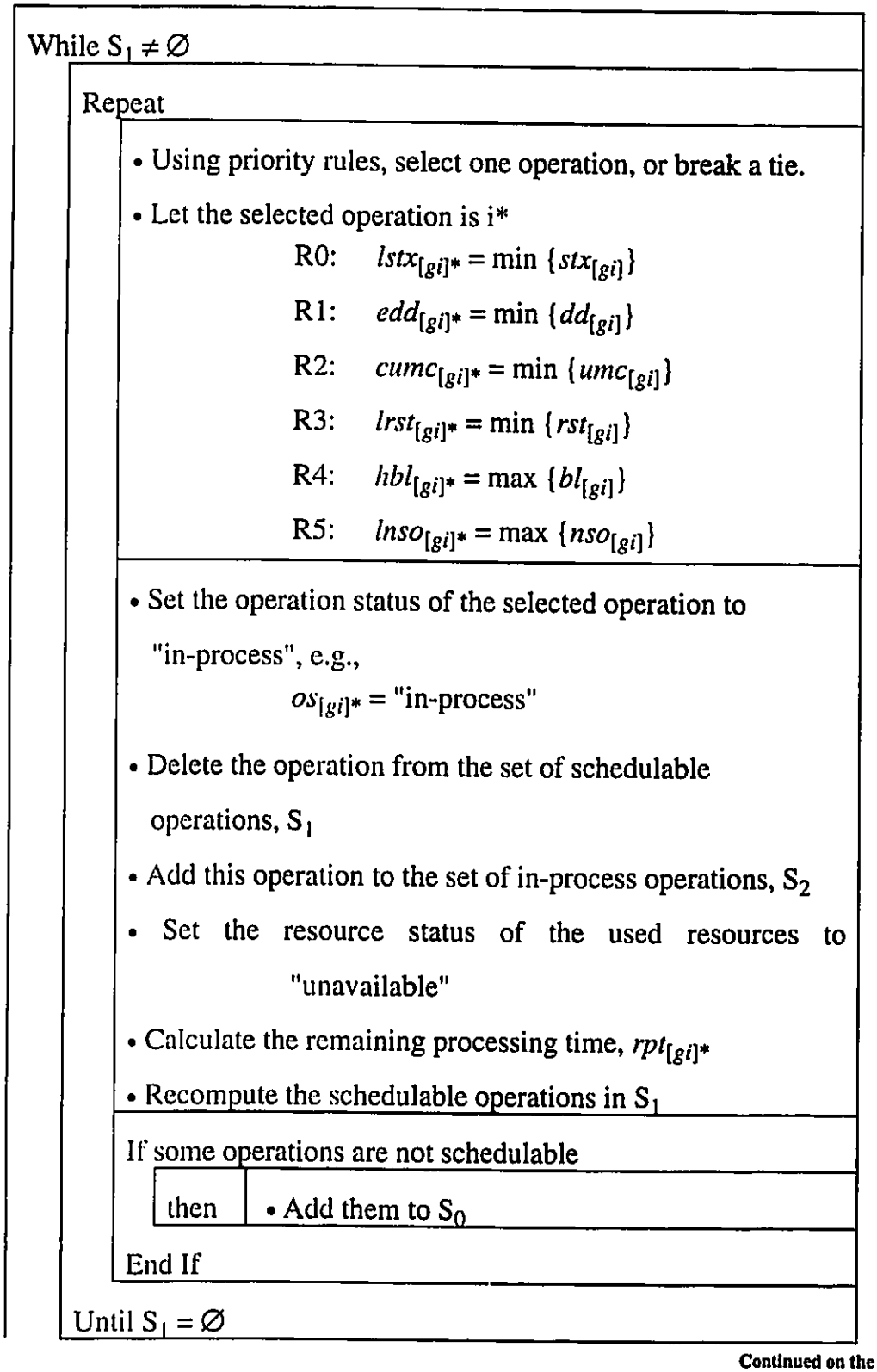
{0: unavailable}

The simulative scheduler heuristic algorithm is depicted in Figures 7.13a, 7.13b and 7.13c.



Continued on the next page...

Figure 7.13a Simulative scheduler heuristic algorithm (Part 1)



Continued on the next page...

Figure 7.13b Simulative scheduler heuristic algorithm (Part 2)

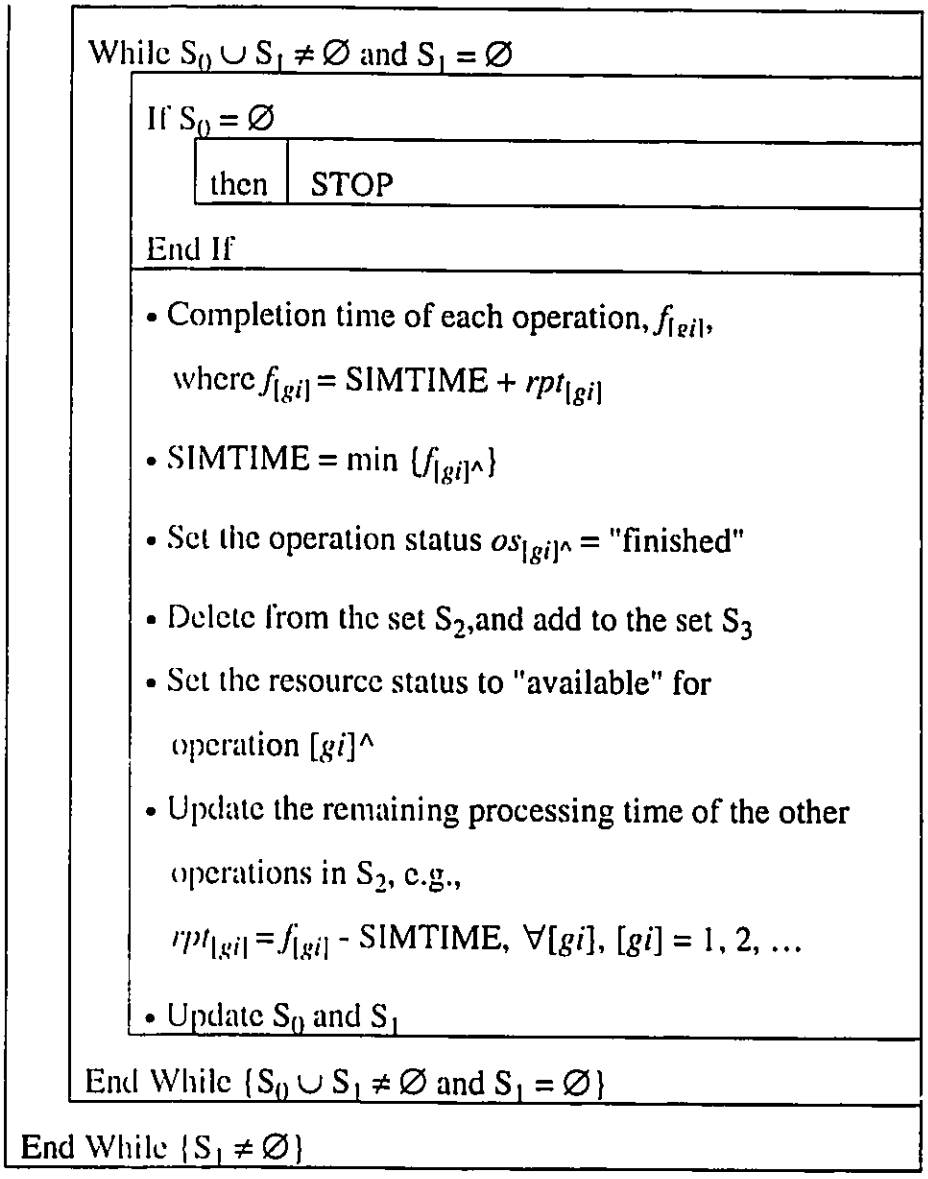


Figure 7.13c Simulative scheduler heuristic algorithm (Part 3)

7.5.1 Numerical example

Prepare a final production schedule of three products indicated in the Figure 7.14 with the data given below.

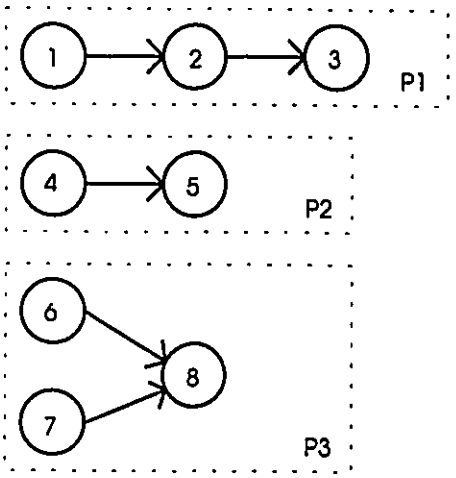


Figure 7.14 Operation flow of products P1, P2, P3

There are three orders in the form

$$\vec{DO} = (O_{ndx}, P_k, Q_{Ondx}, DD_{Ondx}, OAD_{Ondx}, CD_{Ondx})$$

as follows:

$$\vec{DO} = (1, P_1, 1, 17, -, 0);$$

$$\vec{DO} = (2, P_2, 1, 18, -, 0);$$

$$\vec{DO} = (3, P_3, 1, 9, -, 0);$$

The process plans corresponding to each product are in the form

$$\vec{PP} = (PP_{ndx}, DPP_{ndx}, P_k, N\vec{IP}_k, \vec{IP}_k)$$

as follows:

$$\vec{PP} = (1,-, P_1, 3, \{1, 2, 3\});$$

$$\vec{PP} = (2,-, P_2, 2, \{4, 5\});$$

$$\vec{PP} = (3,-, P_3, 3, \{6, 7, 8\});$$

The product information are in the form

$$\vec{P} = (DP_k, P_k, SL_k, PP_{ndx})$$

as follows:

$$\vec{P} = (-, P_1, 0, 1);$$

$\vec{P}$ = (-, P_2, 0, 2);

$\vec{P}$ = (-, P_3, 0, 3);

The operations are in the form

$$\vec{IP}_k = \{(D_i, i, j, isn_j, osn_j, s_i, s_j, t_i, iss_j, oss_j, z_i, w_i, p_i), \ldots$$
$$(\tilde{n}, j, isn_j, osn_j, s_{\tilde{n}}, s_j, t_{\tilde{n}}, iss_j, oss_j, z_{\tilde{n}}, w_{\tilde{n}}, p_{\tilde{n}})\}$$

as follows:

$\vec{IP}_1$ = {(-, 1, 1, -, -, 2.0, 3.0, 2.0, -, -, -, -, -), (-, 2, 3, -, -, 1.0, 1.0, 2.0, -, -, -, -, -),
(-, 3, 2, -, -, 1.0, 1.0, 1.0, -, -, -, -, -)}

$\vec{IP}_2$ = {(-, 4, 1, -, -, 1.0, 1.0, 2.0, -, -, -, -, -), (-, 5, 3, -, -, 1.0, 1.0, 1.0, -, -, -, -, -)}

$\vec{IP}_3$ = {((-, 6, 3, -, -, 1.0, 1.0, 2.0, -, -, -, -, -), (-, 7, 1, -, -, 1.0, 1.0, 2.0, -, -, -, -, -)),
(-, 8, 2, -, -, 1.0, 1.0, 1.0, -, -, -, -, -)}

The data shown in '-' means that are not considered. According to the data, there are three machines. Therefore the resource status of machines can be reset to "available":

$$rs_{1q} = \text{"available" for } \forall q,\ q = 1, 2, 3.$$

In addition, we will use the priority rules of Rule 1, Rule 3, Rule 5 and Rule 6. The solution steps are given in Appendix A1. The final schedule and operation status are given in Figure 7.15 and Table 7.4.

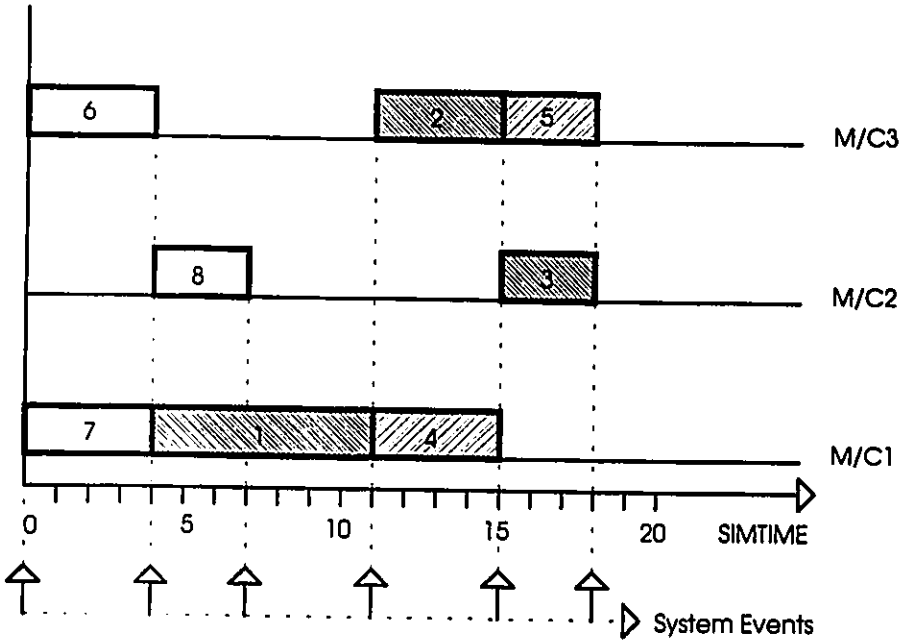


Figure 7.15 Final schedule for the example

Table 7.4 Lateness and tardiness of the operations for the example 7.5.1

Generated Operation Number	Due Date	Completion Time	Lateness	Tardiness
6	9	4	-5	0
7	9	4	-5	0
8	9	7	-2	0
1	17	11	-6	0
2	17	15	-2	0
3	17	18	1	1
4	18	15	-3	0
5	18	18	0	0

8 IMPLEMENTATION RESULTS

Example 1:

Example given in section 7.5.1 is solved with SIMEX. The results are as follows:

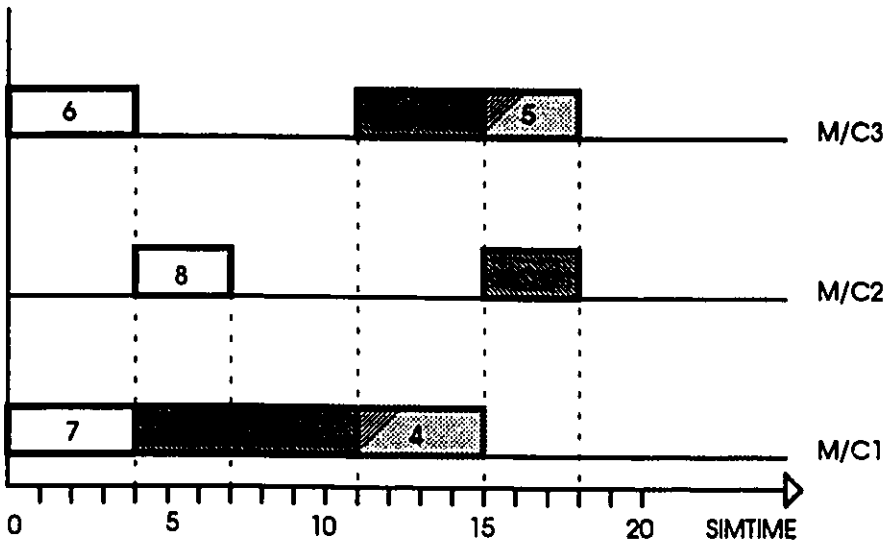


Figure 8.1 Final schedule for the example 7.5.1

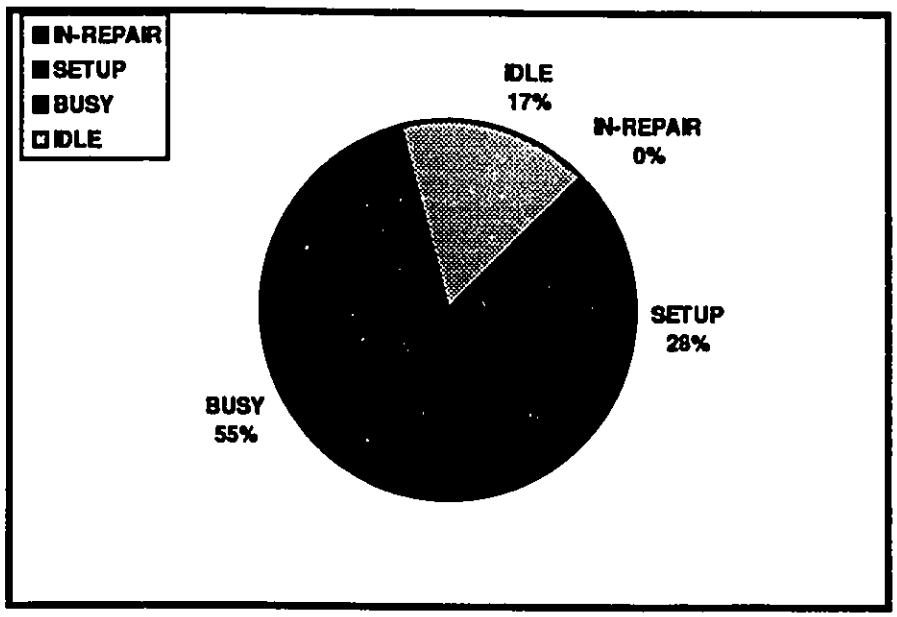


Figure 8.2 Machine 1 (M/C1) utilization measurements

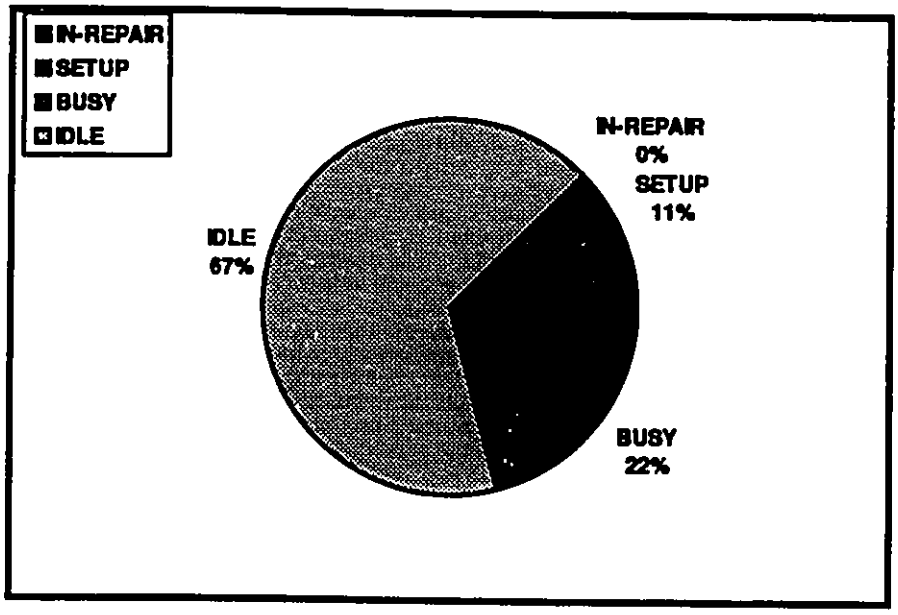


Figure 8.3 Machine 2 (M/C2) utilization measurements

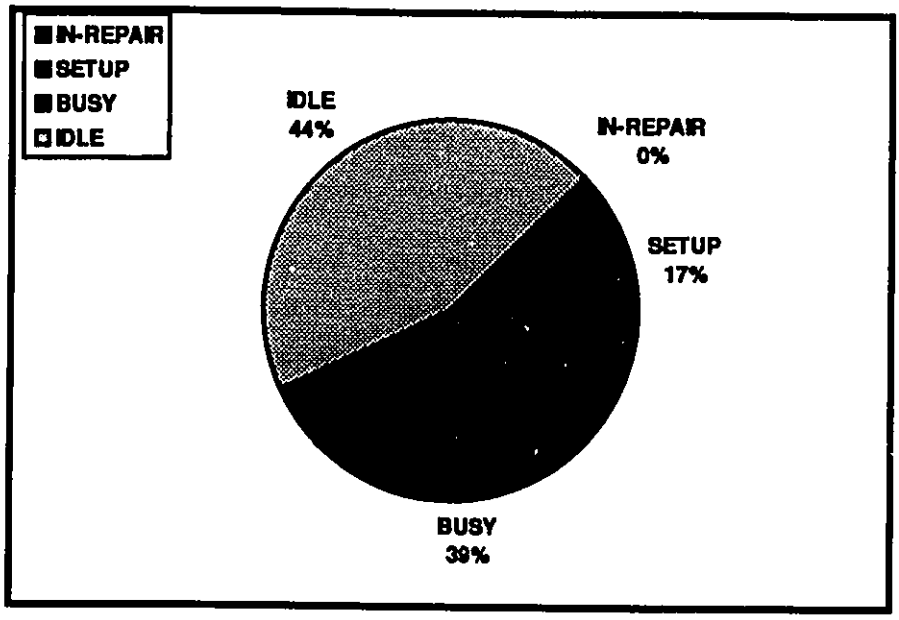


Figure 8.4 Machine 3 (M/C3) utilization measurements

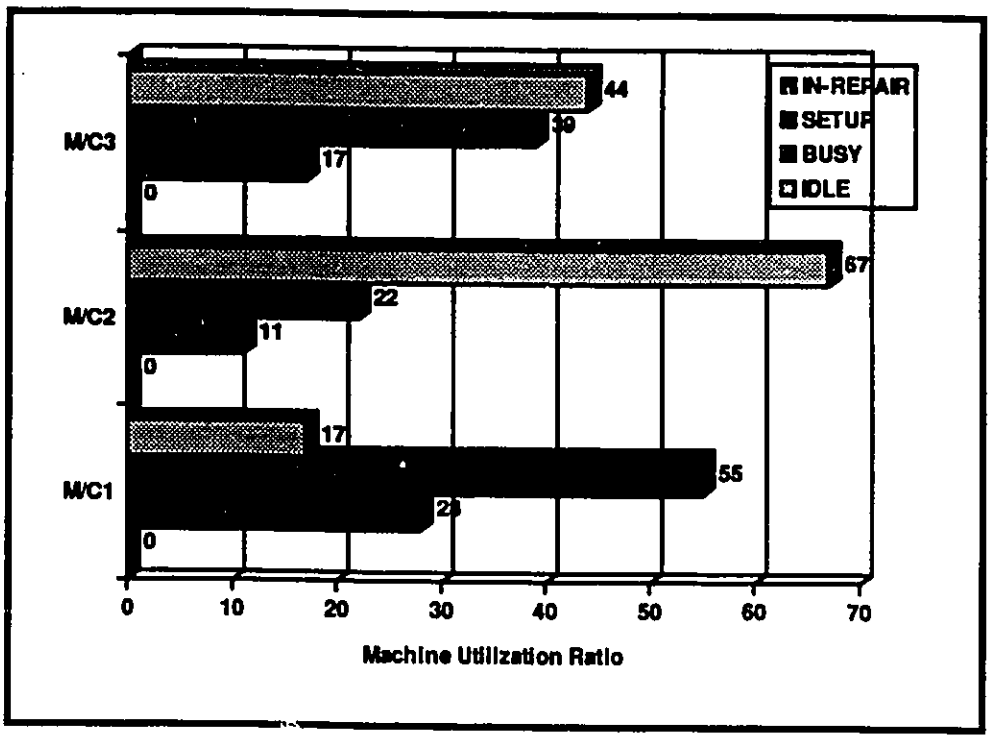


Figure 8.5 Comparison (M/C) utilization measurements

Example 2:

As another scenario, lets say, there is an unexpected event that machine 1 had been reserved for an extra work at SIMTIME = 6.0. And expected work time is 2.0. Then SIMEX generates the following production schedule. The solution steps are given in Appendix A2.

The final schedule for the new scenario is depicted in Figure 8.6, and Table 8.1. And the machine utilization measurements are shown in Figure 8.7 through Figure 8.10.

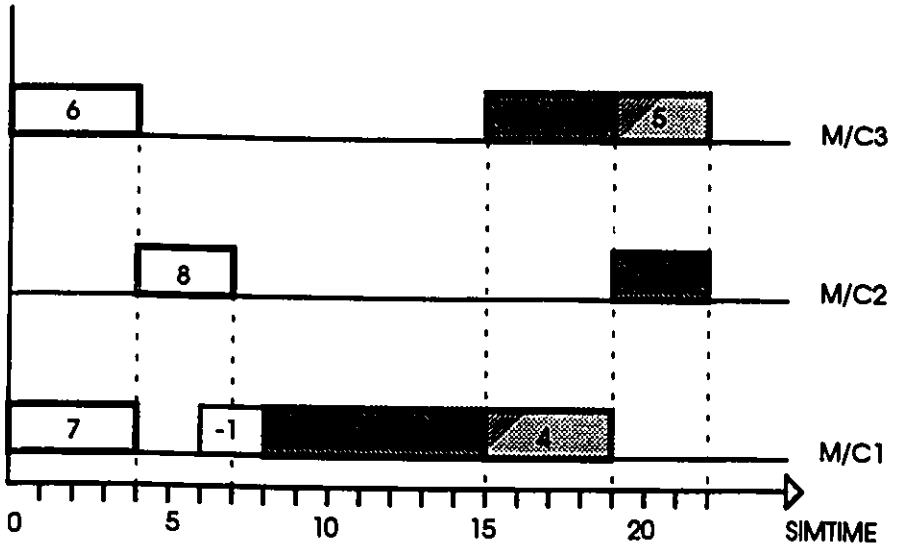


Figure 8.6 Final schedule for the new scenario

Table 8.1 Lateness and tardiness of the operations

Generated Operation Number	Due Date	Completion Time	Lateness	Tardiness
6	9	4	-5	0
7	9	4	-5	0
8	9	7	-2	0
1	17	15	-2	0
2	17	19	2	2
3	17	22	5	3
4	18	19	1	1
5	18	20	2	2

As you can see from Table 8.1, number of late operations is increased from 1 to 4.

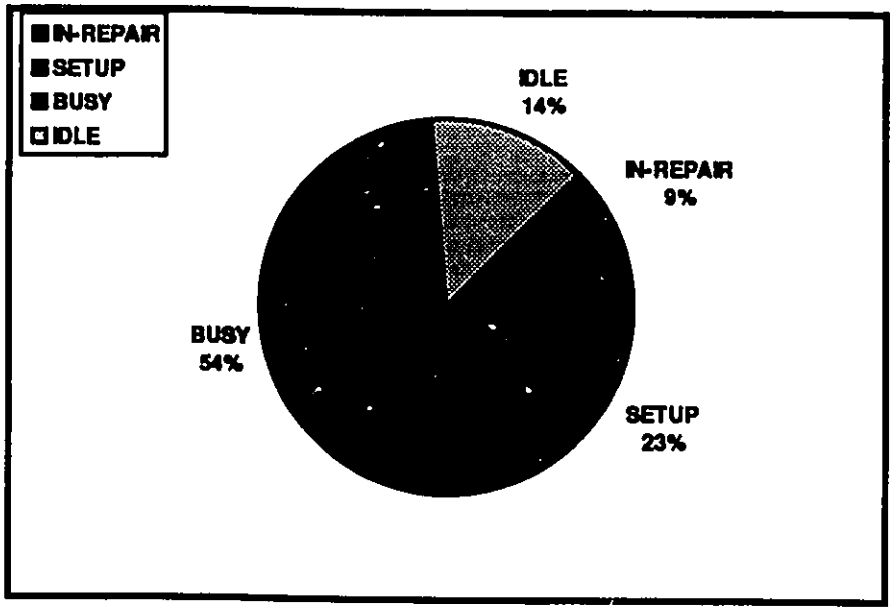


Figure 8.7 Machine 1 (M/C1) utilization measurements

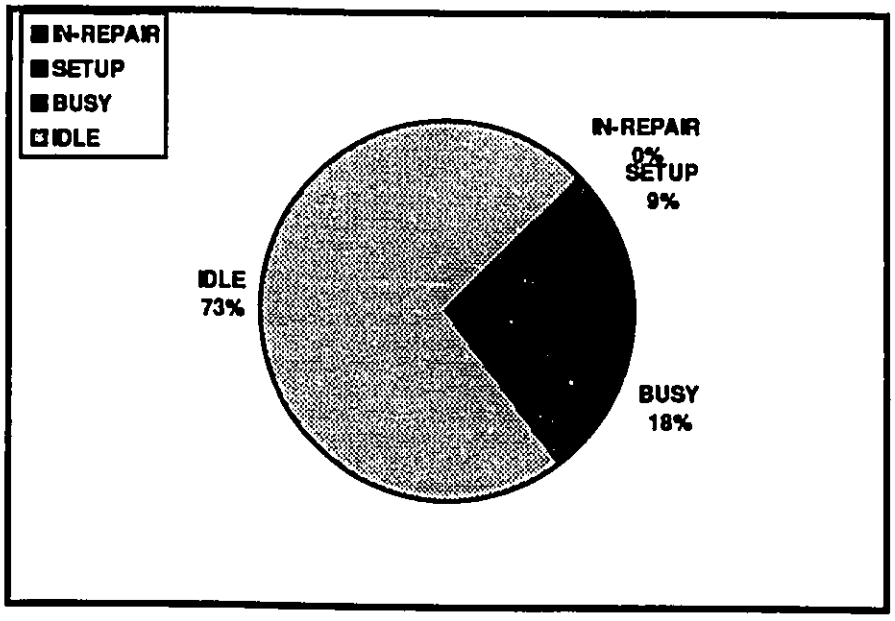


Figure 8.8 Machine 2 (M/C2) utilization measurements

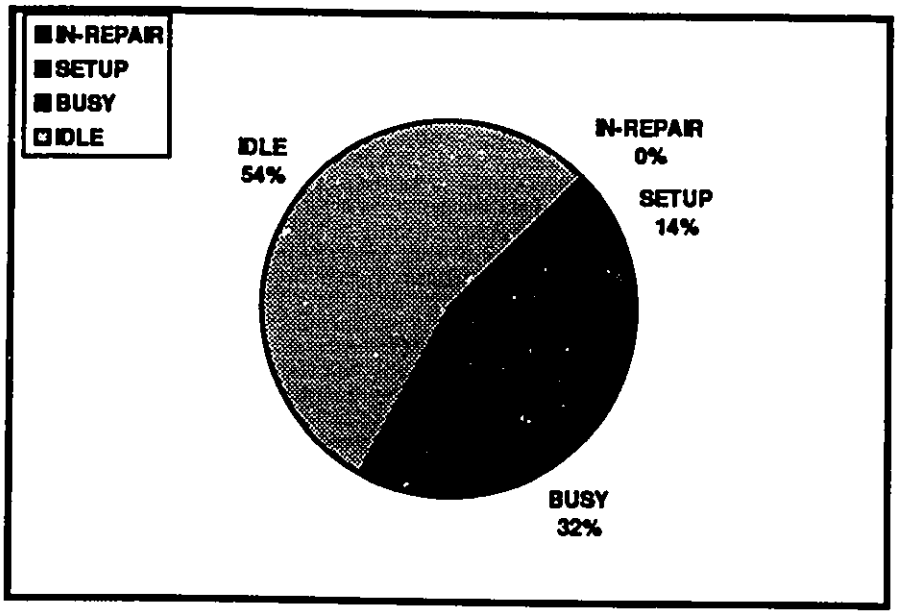


Figure 8.9 Machine 3 (M/C3) utilization measurements

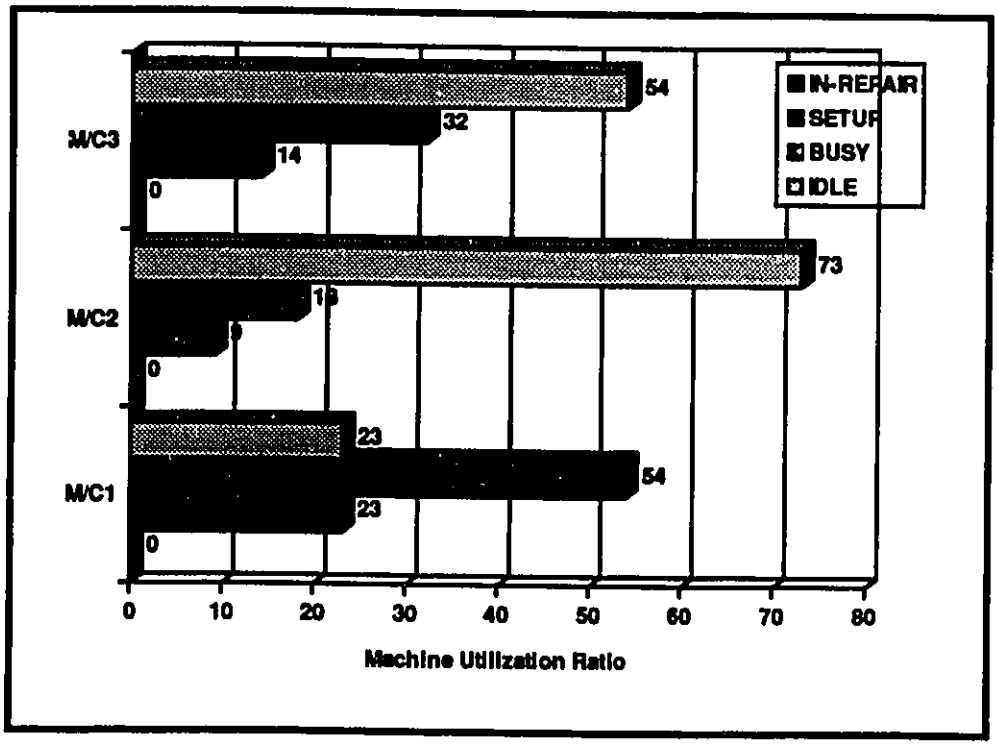


Figure 8.10 Comparison (M/C) utilization measurements

Example 3:

This time, lets say, there is an unexpected event that machine 1 had been broken down at SIMTIME = 6.0. And expected work time is 2.0. Then SIMEX generates the following production schedule. The solution steps are given in Appendix A3.

The final schedule for the new scenario is depicted in Figure 8.11, and Table 8.2. The machine utilization measurements are shown in Figure 8.7 through Figure 8.10.

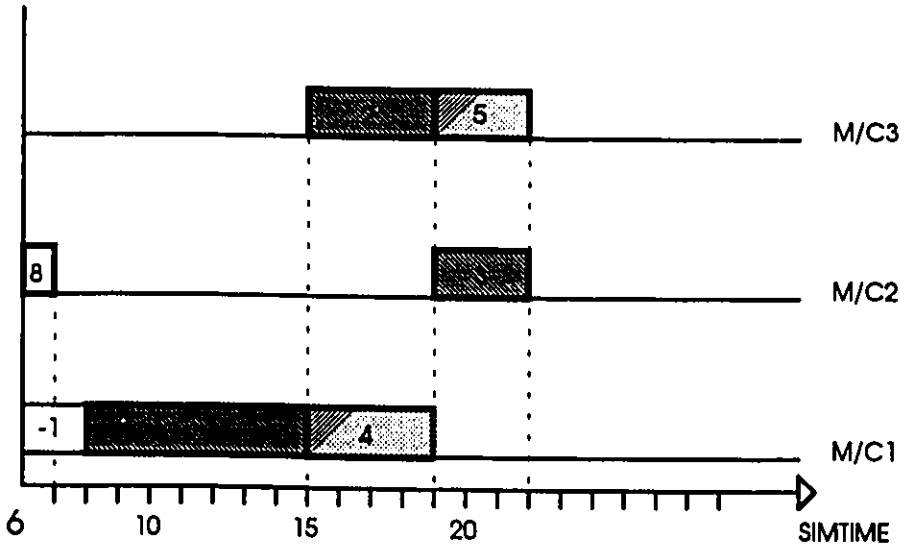


Figure 8.11 Final schedule for the new scenario

Table 8.2. Lateness and tardiness of the operations

Generated Operation Number	Due Date	Completion Time	Lateness	Tardiness
8	9	7	-2	0
1	17	15	-2	0
2	17	19	2	2
3	17	22	5	3
4	18	19	1	1
5	18	20	2	2

As you can see from Table 8.2, number of late operations is 4.

9 CONCLUSIONS AND FUTURE WORK

This study has established two main goals: (1) extending the knowledge on production scheduling problems that require to handle unexpected events in a dynamic manufacturing environment, and (2) developing a computer software that uses the latest available software engineering technology.

From the industrial and manufacturing engineering perspective, the following results are significant: (1) It is desirable to apply scheduling dispatching rules interactively according to the manufacturing conditions, (2) it is desirable to have a software to generate valid production schedules immediately when current schedule is invalidated by an unexpected event, (3) in terms of improved production control of the manufacturing system with generated production schedules, integration of simulation and knowledge-based system technologies is succeeded, and (4) it is highly desirable to have a user friendly interface to display visualized obtained schedules and performance measurements on the screen.

From the software engineering perspective, this study has established the following results: (1) graphical user interface (GUI) programming is succeeded as well as creating and developing an industrial production scheduler. Microsoft Visual Basic is used for creation of user interface to facilitate manipulation of manufacturing events to users, (2) object oriented programming applications are accomplished with MODSIM II and Microsoft Visual C++, (3) artificial intelligence programming by Prolog, thus human expertise, such as rule-of-thumbs, could be included in the production scheduler to increase the precision of the generated schedules, and (4) dynamic data exchanging (DDE) between the applications to succeed immediately displaying schedule results on the screen and continuously communication.

The reasons behind the development of SIMEX can be counted as follows: (1) traditional programming techniques can help only conventional production scheduling models, (2) conventional production scheduling techniques fail to represent a realistic factory model, (3) unexpected events in a dynamic manufacturing environment cannot be manipulated with the conventional techniques, (4) availability of the recent tools for object oriented programming, simulation and knowledge-based systems development, and graphical user interface programming (GUI), and (5) availability of operating systems that enables interprocess communication.

SIMEX, the prototype, can be improved by adding extra features. There would be a heuristic algorithm base among which SIMEX can pick the best algorithm to execute for the current situation. Moreover, group technology among the products can be implemented to achieve reduced production lead time which helps finishing products before due dates. In addition to that, a new feature as possibility of changing the priority for a certain job might be added. It is therefore more control over the job control would be achieved in some critical decisions resulting high performance and high resource utilization.

Learning behavior can be added to the system to generalize experienced problems and their solution path to use later whenever SIMEX experiences again. It would therefore avoid spending time to solve similar problems, instead of again and again repeating same loops and solution paths. Solution paths could be kept in solution libraries for human understanding as well. Whenever an explanation is required by the user, SIMEX could follow pretracked path and give detailed explanation of the action taken.

A valid calendar can be added to control holidays and vacations of human resources. Therefore, more accurate production schedules would be achieved by knowing availability of human operators.

SIMEX can be ported in a UNIX environment, or at least UNIX-like environment to have asynchronous possession to handle unexpected events more precisely in real-time. Asynchronous execution of the program modules would increase the response time whenever unexpected events occur. Therefore SIMEX can be acknowledged by asynchronous communication between the modules. Interprocess communication, background process execution are some of the advantageous of UNIX environment for real-time applications such as SIMEX industrial production scheduler. By having multiple workstations in different departments within an organization, SIMEX can be upgraded and maintained , database can be kept current, and hardware can be upgraded due to the increased sophistication. It might also be integrated to an MRP II system, production can be controlled by SIMEX schedules as well as MRP schedules.

References:

ACSL, (1991). Advanced Continuous Simulation Language (ACSL) Reference Manual, MGA Inc, MA.

Arons, H. de S. (1990). Real-time Expert Scheduling in Manufacturing. In: ESS '90, Intelligent Process Control and Scheduling, Discrete Event Systems, G.C. Vansteenkiste, E.J.G. Kerckhoffs, H. Müller (-Malek), and F. Broeckx (eds.), Society for Computer Simulation International, San Diego, CA, pp. 124-127.

Badiru, A.B. (1990). Artificial Intelligence Applications in Manufacturing. In: The Automated Factory Handbook, Technology and Management, D.I. Cleland and B. Bidanda (eds.), TAB Professional and Reference Handbooks, Blue Ridge Summit, PA, pp. 496-526.

Baker, K.R. (1974). Introduction to Sequencing and Scheduling. John Wiley, New York.

Baker, K.R. (1992). Elements of Sequencing and Scheduling. Amos Tuck School of Business Administration, Dartmouth College, Hanover, NH.

Barnes, T.A. and L.L. Gardner, (1990). A Simulation Model for Production Scheduling of a Facility in a Process Industry. In: 1990 Winter Simulation Conference Proceedings, Balci, O., Sadowski R.P. and Nance, R.E. (eds.) , The Society of Computer Simulation, CA.

Banks, J., E. Aviles, J.R. McLaughlin, and R.C. Yuan, (1991). The Simulator: New Member of the Simulation Family. In: Interfaces, Vol. 21, pp. 76-86.

Ben-Arieh, D. (1986). Knowledge-based Control System for Automated Production and Assembly. In: Modelling and Design of Flexible Manufacturing Systems, A. Kusiak (ed.), Elsevier, New York, pp. 347-368.

Binbasioglu, M. and M. Jarke, (1986). Domain-Specific DSS Tools for Knowledge-Based Model Building. In: Decision Support Systems, Vol. 2, pp. 213-223.

Blackstone, Jr. J.H, D.T. Phillips, and G.L. Hogg. A State-of-the-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations. In: Int. J. Prod. Res. Vol. 20, No. 1, pp 27-45. 1982.

Bratko, I. (1991). Prolog: Programming for Artificial Intelligence (2nd ed). Reading, MA: Addison-Wesley.

Brownston, L., R. Farrel, E. Kant and N, Martin, (1985). Programming Expert Systems in OPS5: An Introduction to Rule-based Programming. Reading, MA: Addison-Wesley.

Bruno, B. et al., (1986). A Rule-based System to Schedule production. In: IEEE Computer, Vol. 19, No. 7, pp. 32-40.

Burns, F. and J. Rooker, (1976). Johnson's Three-Machine Flow Shop Conjecture. In: Operations Research, Vol. 24, pp. 578-580.

Burns, F. and J. Rooker, (1978). Three Stage Flow Shops with Regressive Second Stage. In: Operations Research, Vol. 26, pp. 207-208.

CACI, (1984). An Introduction to Simulation Using Simscript II.5, CACI Products Company, CA.

Campbell, H.G., R.A. Dudek, and M.L. Smith, (1970). A Heuristic Algorithm for the *n* Job, *m* Machine Sequencing Problem. In: Management Science, Vol. 16, pp. 630-637.

De, S. (1988). Knowledge Representation in Manufacturing Systems. In Expert Systems: Strategies and Solutions in Manufacturing Design and Planning, Society for Manufacturing Engineers, Dearborn, Mich., pp. 79-107.

Ebner, M.L. and T.E. Vollman, (1988). Manufacturing Systems for the 1990's. In: Intelligent Manufacturing, M. Oliff (ed.), The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA.

Erschler, J. and P. Esquirol, (1986). Decision-aid in Job Shop Scheduling: A Knowledge-based Approach. In: Proceedings of the 1986 IEEE International Conference on Robotics and Automation, San Francisco, CA, April 7-10, pp. 1651-1656.

Emmons, H. (1969). One-Machine Sequencing to Minimize Certain Functions of Job Tardiness. In: Operations Research, Vol. 17, pp. 701-715.

Fargher, H.E. and R.A. Smith, (1992). Planning for the Semiconductor Manufacturer of the Future. In: Practical Approaches to Scheduling and Planning (Papers from the 1992 Spring Symposium Technical Report SS-92-01). AAAI Press, CA, pp. 57-61.

Fox, M.S. and S.F. Smith, (1984). ISIS: A Knowledge-based System for Factory Scheduling. In: Expert Systems Journal, Vol. 1, No. 1, pp. 25-49.

French, S., (1982). Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop. John Wiley, New York.

Glover, F. (1989). Tabu Search - Part I. In: ORSA Journal on Computing, No. 1, pp. 190-203.

Haider, S.W., and J. Banks, (1986). Simulation Software Products for Analyzing Manufacturing Systems. In: Industrial Engineering, Vol. 18, pp. 98-103.

Harel, D. (1987). Algorithmics: The Spirit of Computing. Addison-Wesley, Massachusetss.

Johnson, S.M., (1954). Optimal two- and three-stage production schedules with set-up times included. In: Naval Research Logistics Quarterly, Vol. 1, pp. 61-68.

Jürgen, D., S. Wolfgang, and S. Christian (1992). Uncertainty Management by Relaxation of Conflicting Constraints in Production Process Scheduling. In: Practical Approaches to Scheduling and Planning, Papers from the 1992 AAAI Spring Symposium, AAAI Press, CA., pp. 62-66.

Kim, S.H., (1991). Knowledge Systems Through Prolog. Oxford University Press, New York 10016.

Kolodner, J. and et al. (1985). A Process of Case-based Reasoning in Problem Solving. In: Proceeding of the Ninth International Joint Conference on Artificial Intelligence. IJCAI, Los Angeles, CA, pp. 284-290.

Kowalski, R. (1979). Logic for Problem Solving, Elsevier, New York.

Kurisu, T., (1976). Two-machine Scheduling Under Required Precedence Among Jobs. In: Operational Research Society (Japan), 19, pp. 1-13.

Kusiak, A., (1986). Efficient Implementation of Johnson's Scheduling Algorithm. In: IIE Transactions, Vol. 18, No. 2, pp. 215-216.

Kusiak, A. (1987). Artificial Intelligence and Operations Research in Flexible Manufacturing Systems. In: Information Processing and Operational Research (INFOR), Vol. 25, No. 1, pp. 2-12.

Kusiak, A., (1990). Intelligent Manufacturing Systems. Prentice Hall International Series in Industrial and Systems Engineering, Englewood Cliffs, NJ.

Kusiak, A., (1991). Planning, Scheduling, and Control of Manufacturing Systems. In: Handbook of Flexible Manufacturing Systems, N.K. Jha (ed.), Academic Press, Inc., London, NW.

Lawler, E.L., (1977). A 'Pseudopolynomial' Algorithm for Sequencing Jobs to Minimize Total Tardiness. In: Annals of Discrete Mathematics, No. 1, pp. 331-342.

Lawler, E.L., (1982). A Fully Polynomial Approximation Scheme for the Total Tardiness Problem. In: Operations Research Letters, No. 1, pp. 207-208.

Luger, G.F. and W.A. Stubblefield, (1989). Artificial Intelligence and the Design of Expert Systems. The Benjamin/Cummings Publishing Company, Inc., Redwood city, CA.

MacFarland, D. and M.E. Grant, (1990). Tutorial: Scheduling Manufacturing Systems With Factor. In: 1990 Winter Simulation Conference Proceedings. O. Balci, R.P. Sadowski, and R.E. Nance (eds.), SCS, CA, pp. 146-148.

Mebarki, N., and et al. (1993). On Modelling Dynamic Scheduling in Object-Oriented Simulation of Flexible Manufacturing Systems. In: Modellling and Simulation ESM 93, Proceedings of the 1993 European Simulation Multiconference. A. Pave (ed.), SCS, CA, pp. 379-383.

Meyer, W., (1990). Expert Systems in Factory Management, Ellis Harwood, West Sussex, England.

Minsky, M., (1975). A Framework for Representing Knowledge. In: P.H. Winston (ed.), The Psychology of Computer Vision, McGraw-Hill, New York, pp. 211-227.

Miyashita, K. and K. Sycara, (1992). CABINS: Case-based Interactive Scheduler. In: Practical Approaches to Scheduling and Planning (Papers from the 1992 Spring Symposium Technical Report SS-92-01). AAAI Press, CA, pp. 47-51.

MODSIM II, (1991). The Language for Object-Oriented Programming, Reference Manual, CACI Products Company, CA.

Moore, J.M., (1968). Sequencing n Jobs on One Machine to Minimize the Number of Tardy Jobs. In: Management Science, No. 15, pp. 102-109.

Morton, T.E. and T.L. Smunt, (1986). A Planning and Scheduling System for Flexible Manufacturing. In: Flexible Manufacturing Systems: Methods and Studies, A. Kusiak (ed.), North Holland, Amsterdam, The Netherlands, pp. 151-164.

Murphy, F.H. and E.A. Stohr, (1986). An Intelligent System for Formulating Linear Programs. In: Decision Support Systems, Vol. 2, pp. 39-47.

Newman, P.A. and K.G. Kempf, (1985). Opportunistic Scheduling for Robotic Machine Tending. In: The 2nd Conference on Artificial Intelligence Applications, Miami Beach, Fla., December 11-13, pp. 168-173.

Niwa, K., K. Sasaki, and K. Ihara, (1984). An Experimental Comparison of Knowledge Representation Schemes. In: The AI Magazine, Vol. 5, No. 2, pp. 29-36.

Ören, T. I. (1984). Graphic Representation of Pseudocodes and Computer Programs: A Unifying Technique and a Family of Documentation of Programs. In: Proc. of EdCompCon-83 (First Educational Computing Conf. of IEEE Computer Society), D.C. Rine (ed.). San Jose, CA., Oct. 18-20, 1983. IEEE Computer Society, New York, pp. 81-89.

Potts, C.N. and L.N. Van Wassenhove, (1982). A Decomposition Algorithm for the Single Machine Total Tardiness Problem. In: Operations Research Letters, No. 1, pp. 177-181.

Sadeh, N.M. (1992). The MICRO-BOSS Scheduling System: Current Status and Future Efforts. In: Practical Approaches to Scheduling and Planning (Papers from the 1992 Spring Symposium Technical Report SS-92-01). AAAI Press, CA, pp. 37-41.

Sedgewick, R. (1988). Algorithms, Addison-Wesley, Massachusetts.

Shannon, R.F., (1975). Simulation: A Survey with Research Suggestions. In: AIEE Transactions, Vol. 7, No. 3, September.

Shwimer, J., (1972). On the n-Job, One-Machine, Sequence Independent Scheduling Problem with Tardiness Penalties: A Branch and Bound Solution. In: Management Science, No. 18, pp. 301-313.

Smith, S.F. (1988). A Constraint-based Framework for Reactive Management of Factory Schedules. In: Intelligent Manufacturing, M. Oliff (ed.), The Benjamin/Cummings Publishing Company, Inc., Menlo Park, CA.

Szwarc, W., (1977). Optimal Two Machine Orderings in the 3x*n* Flow Shop Problem. In: Management Science, Vol. 25, pp. 70-77.

Winston, W.L. (1991). Operations Research Applications and Algorithms, PWS-KENT, Boston, Massachusetts.

Wyman, F.P. (1991). Common Features of Simulation Based Scheduling. In: Proceedings of the 1991 Winter Simulation Conference. B.L. Nelson, W.D. Kelton, and G. M. Clark (eds.), SCS, CA, pp. 341-347.

APPENDICES

A.1 Solution Steps of Example 7.5.1 (pp. 105)

If you apply the simulative scheduler algorithm, the following solution procedure will be obtained. Each iteration shows one step in the heuristic algorithm.

Iteration[1] Set SIMTIME = 0.0;

{ set all machines to "available"}

rs_{11} = "available";

rs_{12} = "available";

rs_{13} = "available";

Iteration[2] $S_0 = \{2, 3, 5, 8\}$;

$S_1 = \{1, 4, 6, 7\}$;

$S_2 = \varnothing$;

$S_3 = \varnothing$;

$S_e = \varnothing$;

Iteration[3] Rule 1 $\Rightarrow \{6, 7\}$

Rule 3 $\Rightarrow \{6,7\}$

Rule 5 $\Rightarrow \{6,7\}$

Rule 6 $\Rightarrow \{6\}$

$\therefore$Operation 6* is selected.

Iteration[4] $os_{[6]*}$ = "in-process";

$S_1 = \{1, 4, 7\}$;

$S_2 = \{6\}$;

rs_{13} = "unavailable";

$rpt_{[6]^*} = 4.0$;

$S_1 = \{1, 4, 7\}$;

$S_0 = \{2, 3, 5, 8\}$;

$S_1 \neq \varnothing$ then go to step 2;

Iteration[5] Rule 1 $\Rightarrow \{7\}$

$\therefore$ Operation 7* is selected.

Iteration[6] $os_{[7]^*}$ = "in-process";

$S_1 = \{1, 4\}$;

$S_2 = \{7, 6\}$;

rs_{11} = "unavailable";

$rpt_{[7]^*} = 4.0$;

$S_1 = \varnothing$;

$S_0 = \{1, 2, 3, 4, 5, 8\}$;

$S_1 = \varnothing$ then go to step 4;

Iteration[7] {Simulation time plus remaining operation time}

$f_{[6]^*}$ = SIMTIME + 4.0 = 4.0;

$f_{[7]^*}$ = SIMTIME + 4.0 = 4.0;

SIMTIME = SIMTIME + min $\{f_{[6]^*}, f_{[7]^*}\} = 4.0$;

$os_{[6]^*}$ = "finished";

$os_{[7]^*}$ = "finished";

$S_2 = \varnothing$;

$S_3 = \{7,6\}$;

rs_{13} = "available";

rs_{11} = "available";

Iteration[8] $S_0 = \{2, 3, 5\}$;

$S_1 = \{1, 4, 8\}$;

Final Rule 3 $\Rightarrow$ go to step 2;

Iteration[9] Rule 1 $\Rightarrow \{8\}$

$\therefore$ Operation 8* is selected.

Iteration[10] $os_{[8]^*}$ = "in-process";

$S_1 = \{1, 4\}$;

$S_2 = \{8\}$;

rs_{12} = "unavailable";

$rpt_{[8]^*} = 3.0$;

$S_1 = \{1, 4\}$;

$S_0 = \{2, 3, 5\}$;

$S_1 \neq \varnothing$ then go to step 2;

Iteration[11] Rule 1 $\Rightarrow \{1\}$

$\therefore$ Operation 1* is selected.

Iteration[12] $os_{[1]^*}$ = "in-process";

$S_1 = \{4\}$;

$S_2 = \{1, 8\}$;

rs_{11} = "unavailable";

$rpt_{[1]^*} = 7.0$;

$S_1 = \varnothing$;

$S_0 = \{2, 3, 4, 5\}$;

$S_1 = \varnothing$ then go to step 4;

Iteration[13] {Simulation time plus remaining operation time}

$f_{[8]*}$ = SIMTIME + 3.0 = 7.0;

$f_{[1]*}$ = SIMTIME + 7.0 = 11.0;

SIMTIME = SIMTIME + min $\{f_{[8]*}, f_{[1]*}\}$ = 7.0;

$os_{[8]*}$ = "finished";

$S_2 = \{1\}$;

$S_3 = \{8, 7, 6\}$;

rs_{12} = "available";

$rpt_{[1]*} = f_{[1]*}$ - SIMTIME = 11.0 - 7.0 = 4.0;

Iteration[14] $S_0 = \{2, 3, 4, 5\}$;

$S_1 = \varnothing$;

Final Rule 2 $\Rightarrow$ go to step 4;

Iteration[15] {Simulation time plus remaining operation time}

$f_{[1]*}$ = SIMTIME + 4.0 = 11.0;

SIMTIME = min $\{f_{[1]*}\}$ = 11.0;

$os_{[1]*}$ = "finished";

$S_2 = \varnothing$;

$S_3 = \{1, 8, 7, 6\}$;

rs_{11} = "available";

Iteration[16] $S_0 = \{3, 5\}$;

$S_1 = \{2, 4\}$;

Final Rule 3 $\Rightarrow$ go to step 2;

Iteration[17] Rule 1 $\Rightarrow$ {2}

$\therefore$Operation 2* is selected.

Iteration[18] $os_{[2]*}$ = "in-process";

$S_1 = \{4\}$;

$S_2 = \{2\}$;

rs_{13} = "unavailable";

$rpt_{[2]*} = 4.0$;

$S_1 = \{4\}$;

$S_1 \neq \varnothing$ then go to step 2;

Iteration[19] Rule 1 $\Rightarrow$ {4}

$\therefore$Operation 4* is selected.

Iteration[20] $os_{[4]*}$ = "in-process";

$S_1 = \varnothing$;

$S_2 = \{4, 2\}$;

rs_{11} = "unavailable";

$rpt_{[4]*} = 4.0$;

$S_1 = \varnothing$ then go to step 4;

Iteration[21] {Simulation time plus remaining operation time}

$f_{[2]*}$ = SIMTIME + 4.0 = 15.0;

$f_{[4]*}$ = SIMTIME + 4.0 = 15.0;

SIMTIME = min $\{f_{[4]*}, f_{[2]*}\}$ = 15.0;

$os_{[2]*}$ = "finished";

$os_{[4]^*}$ = "finished";

$S_2 = \varnothing$;

$S_3 = \{2, 4, 1, 8, 7, 6\}$;

rs_{13} = "available";

rs_{11} = "available";

Iteration[22] $S_0 = \varnothing$;

$S_1 = \{3,5\}$;

Final Rule 3 $\Rightarrow$ go to step 2;

Iteration[23] Rule 1 $\Rightarrow \{3\}$

$\therefore$ Operation 3* is selected.

Iteration[24] $os_{[3]^*}$ = "in-process";

$S_1 = \{5\}$;

$S_2 = \{3\}$;

rs_{12} = "unavailable";

$rpt_{[3]^*} = 3.0$;

$S_1 = \{5\}$;

$S_1 \neq \varnothing$ then go to step 2;

Iteration[25] Rule 1 $\Rightarrow \{5\}$

$\therefore$ Operation 5* is selected.

Iteration[26] $os_{[5]^*}$ = "in-process";

$S_1 = \varnothing$;

$S_2 = \{5, 3\}$;

rs_{13} = "unavailable";

$rpt_{[5]*}$ = 3.0;

$S_1 = \varnothing$ then go to step 4;

Iteration[27] {Simulation time plus remaining operation time}

$f_{[3]*}$ = SIMTIME + 3.0 = 18.0;

$f_{[5]*}$ = SIMTIME + 3.0 = 18.0;

SIMTIME = min { $f_{[5]*}$, $f_{[3]*}$ } = 18.0;

$os_{[3]*}$ = "finished";

$os_{[5]*}$ = "finished";

$S_2 = \varnothing$;

S_3 = {5, 3, 2, 4, 1, 8, 7, 6};

rs_{12} = "available";

rs_{13} = "available";

Iteration[28] $S_0 = \varnothing$;

$S_1 = \varnothing$;

Final Rule 1 $\Rightarrow$ STOP;

A.2 Solution Steps of Example 2 (pp. 111)

If you apply the simulative scheduler algorithm, the following solution procedure will be obtained.

Priority Rules 0, 1, 3, 5, 6 will be used.

Iteration[1] Set SIMTIME = 0.0;

{ set all machines to "available"}

rs_{11} = "available";

rs_{12} = "available";

rs_{13} = "available";

Iteration[2] $S_0 = \{2, 3, 5, 8\}$;

$S_1 = \{1, 4, 6, 7\}$;

$S_2 = \varnothing$;

$S_e = \{-1\}$;

By Rule 0 event -1 is selected.

The unexpected event will be perceived by SIMEX in the form

$\vec{XE} = (ED_e, e, j, isn_j, osn_j, s_e, s_j, t_e, st_e, ft_e, iss_j, oss_j, z_e, w_e, p_e)$

and the solid schedule for the event is:

$\vec{XE}$ = (-, -1, 1, -, -, -, -, 2.0, 6.0, 8.0, -, -, -, -, -)

which means rs_{11} will be set the status of "unavailable" during the interval SIMTIME [6.0, 8.0].

$S_3 = \{-1\}$

Iteration[3] Rule 1 $\Rightarrow \{6, 7\}$

Rule 3 $\Rightarrow$ {6,7}

Rule 5 $\Rightarrow$ {6,7}

Rule 6 $\Rightarrow$ {6}

$\therefore$ Operation 6* is selected.

Iteration[4] $os_{[6]^*}$ = "in-process";

$S_1 = \{1, 4, 7\}$;

$S_2 = \{6\}$;

rs_{13} = "unavailable";

$rpt_{[6]^*} = 4.0$;

$S_1 = \{-1, 1, 4, 7\}$;

$S_0 = \{2, 3, 5, 8\}$;

$S_1 \neq \varnothing$ then go to step 2;

Iteration[5] Rule 1 $\Rightarrow$ {7}

$\therefore$ Operation 7* is selected.

Iteration[6] $os_{[7]^*}$ = "in-process";

$S_1 = \{1, 4\}$;

$S_2 = \{7, 6\}$;

rs_{11} = "unavailable";

$rpt_{[7]^*} = 4.0$;

$S_1 = \varnothing$;

$S_0 = \{1, 2, 3, 4, 5, 8\}$;

$S_1 = \varnothing$ then go to step 4;

Iteration[7] {Simulation time plus remaining operation time}

$f_{[6]^*}$ = SIMTIME + 4.0 = 4.0;

$f_{[7]^*}$ = SIMTIME + 4.0 = 4.0;

SIMTIME = SIMTIME + min $\{f_{[6]^*}, f_{[7]^*}\}$ = 4.0;

$os_{[6]^*}$ = "finished";

$os_{[7]^*}$ = "finished";

$S_2 = \varnothing$;

$S_3 = \{7, 6, -1\}$;

rs_{13} = "available";

rs_{11} = "available";

Iteration[8] $S_0 = \{2, 3, 5\}$;

$S_1 = \{1, 4, 8\}$;

Final Rule 3 $\Rightarrow$ go to step 2;

Iteration[9] Rule 1 $\Rightarrow \{8\}$

$\therefore$Operation 8* is selected.

Iteration[10] $os_{[8]^*}$ = "in-process";

$S_1 = \{1, 4\}$;

$S_2 = \{8\}$;

rs_{12} = "unavailable";

$rpt_{[8]^*} = 3.0$;

$S_1 = \{1, 4\}$;

$S_0 = \{2, 3, 5\}$;

$S_1 \neq \varnothing$ then go to step 2;

Iteration[11] Rule 1 $\Rightarrow \{1\}$

$\therefore$ Operation 1* is selected.

Iteration[12] $os_{[1]^*}$ = "in-process";

$S_1 = \{4\}$;

$S_2 = \{1, 8\}$;

rs_{11} = "unavailable";

$rpt_{[1]^*} = 7.0$;

Detection of a solid schedule, then schedule to the end.

$S_1 = \varnothing$;

$S_0 = \{2, 3, 4, 5\}$;

$S_1 = \varnothing$ then go to step 4;

Iteration[13] {Simulation time plus remaining operation time}

$f_{[8]^*}$ = SIMTIME + 3.0 = 7.0;

$f_{[1]^*} = ft_{-1} + 7.0 = 15.0$;

SIMTIME = SIMTIME + min $\{f_{[8]^*}, f_{[1]^*}\}$ = 7.0;

$os_{[8]^*}$ = "finished";

$S_2 = \{1\}$;

$S_3 = \{8, 7, 6, -1\}$;

rs_{12} = "available";

$rpt_{[1]^*} = f_{[1]^*}$ - SIMTIME = 15.0 - 7.0 = 8.0;

Iteration[14] $S_0 = \{2, 3, 4, 5\}$;

$S_1 = \varnothing$;

Final Rule 2 $\Rightarrow$ go to step 4;

Iteration[15] {Simulation time plus remaining operation time}

$f_{[1]^*}$ = SIMTIME + 8.0 = 15.0;

SIMTIME = min { $f_{[1]^*}$} = 15.0;

$os_{[1]^*}$ = "finished";

$S_2 = \varnothing$;

S_3 = {1, 8, 7, 6, -1};

rs_{11} = "available";

Iteration[16] S_0 = {3, 5};

S_1 = {2, 4};

Final Rule 3 $\Rightarrow$ go to step 2;

Iteration[17] Rule 1 $\Rightarrow$ {2}

$\therefore$ Operation 2* is selected.

Iteration[18] $os_{[2]^*}$ = "in-process";

S_1 = {4};

S_2 = {2};

rs_{13} = "unavailable";

$rpt_{[2]^*}$ = 4.0;

S_1 = {4};

$S_1 \neq \varnothing$ then go to step 2;

Iteration[19] Rule 1 $\Rightarrow$ {4}

$\therefore$ Operation 4* is selected.

Iteration[20] $os_{[4]^*}$ = "in-process";

$S_1 = \varnothing$;

$S_2 = \{4, 2\}$;

rs_{11} = "unavailable";

$rpt_{[4]^*} = 4.0$;

$S_1 = \varnothing$ then go to step 4;

Iteration[21] {Simulation time plus remaining operation time}

$f_{[2]^*}$ = SIMTIME + 4.0 = 19.0;

$f_{[4]^*}$ = SIMTIME + 4.0 = 19.0;

SIMTIME = min $\{ f_{[4]^*}, f_{[2]^*} \}$ = 19.0;

$os_{[2]^*}$ = "finished";

$os_{[4]^*}$ = "finished";

$S_2 = \varnothing$;

$S_3 = \{2, 4, 1, 8, 7, 6, -1\}$;

rs_{13} = "available";

rs_{11} = "available";

Iteration[22] $S_0 = \varnothing$;

$S_1 = \{3,5\}$;

Final Rule 3 $\Rightarrow$ go to step 2;

Iteration[23] Rule 1 $\Rightarrow \{3\}$

$\therefore$Operation 3* is selected.

Iteration[24] $os_{[3]^*}$ = "in-process";

$S_1 = \{5\}$;

$S_2 = \{3\}$;

rs_{12} = "unavailable";

$rpt_{[3]*} = 3.0$;

$S_1 = \{5\}$;

$S_1 \neq \varnothing$ then go to step 2;

Iteration[25] Rule 1 $\Rightarrow \{5\}$

$\therefore$ Operation 5* is selected.

Iteration[26] $os_{[5]*}$ = "in-process";

$S_1 = \varnothing$;

$S_2 = \{5, 3\}$;

rs_{13} = "unavailable";

$rpt_{[5]*} = 3.0$;

$S_1 = \varnothing$ then go to step 4;

Iteration[27] {Simulation time plus remaining operation time}

$f_{[3]*}$ = SIMTIME + 3.0 = 22.0;

$f_{[5]*}$ = SIMTIME + 3.0 = 22.0;

SIMTIME = min $\{f_{[5]*}, f_{[3]*}\}$ = 22.0;

$os_{[3]*}$ = "finished";

$os_{[5]*}$ = "finished";

$S_2 = \varnothing$;

$S_3 = \{5, 3, 2, 4, 1, 8, 7, 6, -1\}$;

rs_{12} = "available";

rs_{13} = "available";

Iteration[28] $S_0 = \varnothing$;

$S_1 = \varnothing$;

Final Rule 1 ⇒ STOP;

A.3 Solution Steps of Example 3 (pp.115)

If you apply the simulative scheduler algorithm, the following solution procedure will be obtained.

Priority Rules 0, 1, 3, 5, 6 will be used.

Iteration[1] Set SIMTIME = 6.0;

{ set all machines to "available"}

rs_{11} = "available";

rs_{12} = "available";

rs_{13} = "available";

Iteration[2] $S_0 = \{2, 3, 5, 8\}$;

$S_1 = \{1, 4\}$;

$S_2 = \{8\}$;

$S_e = \{-1\}$;

By Rule 0 event -1 is selected.

The unexpected event will be perceived by SIMEX in the form

$\vec{XE} = (ED_e, e, j, isn_j, osn_j, s_e, s_j, t_e, st_e, ft_e, iss_j, oss_j, z_e, w_e, p_e)$ and the solid schedule for the event is:

$\vec{XE}$ = (-, -1, 1, -, -, -, -, 2.0, 6.0, 8.0, -, -, -, -, -)

which means rs_{11} will be set the status of "unavailable" during the interval SIMTIME [6.0, 8.0].

$S_3 = \{-1, 6, 7\}$

Iteration[3] Rule 1 $\Rightarrow \{1\}$

$\therefore$ Operation 1* is selected.

Iteration[4] $os_{[1]*}$ = "in-process";

$S_1 = \{4\}$;

$S_2 = \{1, 8\}$;

rs_{11} = "unavailable";

$rpt_{[1]*} = 7.0$;

Detection of a solid schedule, then schedule to the end.

$S_1 = \varnothing$;

$S_0 = \{2, 3, 4, 5\}$;

$S_1 = \varnothing$ then go to step 4;

Iteration[5] {Simulation time plus remaining operation time}

$f_{[8]*}$ = SIMTIME + 3.0 = 7.0;

$f_{[1]*} = ft_{-1} + 7.0 = 15.0$;

SIMTIME = SIMTIME + min $\{f_{[8]*}, f_{[1]*}\}$ = 7.0;

$os_{[8]*}$ = "finished";

$S_2 = \{1\}$;

$S_3 = \{8, 7, 6, -1\}$;

rs_{12} = "available";

$rpt_{[1]*} = f_{[1]*}$ - SIMTIME = 15.0 - 7.0 = 8.0;

Iteration[6] $S_0 = \{2, 3, 4, 5\}$;

$S_1 = \varnothing$;

Final Rule 2 $\Rightarrow$ go to step 4;

Iteration[7] {Simulation time plus remaining operation time}

$f_{[1]*}$ = SIMTIME + 8.0 = 15.0;

SIMTIME = min { $f_{[1]*}$} = 15.0;

$os_{[1]*}$ = "finished";

$S_2 = \varnothing$;

S_3 = {1, 8, 7, 6, -1};

rs_{11} = "available";

Iteration[8] S_0 = {3, 5};

S_1 = {2, 4};

Final Rule 3 $\Rightarrow$ go to step 2;

Iteration[9] Rule 1 $\Rightarrow$ {2}

$\therefore$Operation 2* is selected.

Iteration[10] $os_{[2]*}$ = "in-process";

S_1 = {4};

S_2 = {2};

rs_{13} = "unavailable";

$rpt_{[2]*}$ = 4.0;

S_1 = {4};

$S_1 \neq \varnothing$ then go to step 2;

Iteration[11] Rule 1 $\Rightarrow$ {4}

$\therefore$Operation 4* is selected.

Iteration[12] $os_{[4]*}$ = "in-process";

$S_1 = \varnothing$;

$S_2 = \{4, 2\}$;

rs_{11} = "unavailable";

$rpt_{[4]^*} = 4.0$;

$S_1 = \varnothing$ then go to step 4;

Iteration[13] {Simulation time plus remaining operation time}

$f_{[2]^*}$ = SIMTIME + 4.0 = 19.0;

$f_{[4]^*}$ = SIMTIME + 4.0 = 19.0;

SIMTIME = min $\{f_{[4]^*}, f_{[2]^*}\}$ = 19.0;

$os_{[2]^*}$ = "finished";

$os_{[4]^*}$ = "finished";

$S_2 = \varnothing$;

$S_3 = \{2, 4, 1, 8, 7, 6, -1\}$;

rs_{13} = "available";

rs_{11} = "available";

Iteration[14] $S_0 = \varnothing$;

$S_1 = \{3,5\}$;

Final Rule 3 $\Rightarrow$ go to step 2;

Iteration[15] Rule 1 $\Rightarrow \{3\}$

$\therefore$ Operation 3* is selected.

Iteration[16] $os_{[3]^*}$ = "in-process";

$S_1 = \{5\}$;

$S_2 = \{3\}$;

rs_{12} = "unavailable";

$rpt_{[3]*} = 3.0$;

$S_1 = \{5\}$;

$S_1 \neq \varnothing$ then go to step 2;

Iteration[17] Rule 1 $\Rightarrow \{5\}$

$\therefore$Operation 5* is selected.

Iteration[18] $os_{[5]*}$ = "in-process";

$S_1 = \varnothing$;

$S_2 = \{5, 3\}$;

rs_{13} = "unavailable";

$rpt_{[5]*} = 3.0$;

$S_1 = \varnothing$ then go to step 4;

Iteration[19] {Simulation time plus remaining operation time}

$f_{[3]*}$ = SIMTIME + 3.0 = 22.0;

$f_{[5]*}$ = SIMTIME + 3.0 = 22.0;

SIMTIME = min $\{f_{[5]*}, f_{[3]*}\}$ = 22.0;

$os_{[3]*}$ = "finished";

$os_{[5]*}$ = "finished";

$S_2 = \varnothing$;

$S_3 = \{5, 3, 2, 4, 1, 8, 7, 6, -1\}$;

rs_{12} = "available";

rs_{13} = "available";

Iteration[20] $S_0 = \varnothing$;

$S_1 = \varnothing$;

Final Rule 1 $\Rightarrow$ STOP;