# A Machine Utilization Analysis Tool

by

Johnson Cheah-Shin Tan

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

Bachelor of Science

and

Master of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1995





Author .....
Department of Electrical Engineering and Computer Science
May 19, 1995

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

Certified by .....
Vladimir Drobny
Company Official
Thesis Supervisor

Accepted by .....
F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

# A Machine Utilization Analysis Tool

by

Johnson Cheah-Shin Tan

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 1995, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science
and
Master of Science in Electrical Engineering

## Abstract

In this thesis, a model is formulated to determine the feasibility of a release schedule. Various problem reduction techniques such as aggregation, selected focusing, and relaxation are developed and carried out to make the problem solvable. In addition, relaxing the problem results in a stochastic interpretation of the formulation. Furthermore, a tool based on this model is created as part of the production analysis tool set available on the CIM (Computer Integrated Manufacturing) system called CAFE (Computer Aided Fabrication Environment).

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

Thesis Supervisor: Vladimir Drobny
Title: Company Official

# Acknowledgments

I would first like to thank Scott Parker of Tektronix for opening my eyes to the wonderful albeit crazy world of manufacturing. Jim Humphrey, another Tektronix engineer, was also instrumental in guiding me on manufacturing issues during my time at Tektronix as well as being a great friend. I'm also grateful to Erik Birkeland and Jon Schultz at Maxim, who despite being overloaded with work always took the time to help me. I would also like to express my gratitude to Vladimir Drobney, my company supervisor, for all his help and for allowing me to stay at his fab working on my thesis.

I would also like to say that I was very fortunate for Dr. Stan Gershwin to have accepted me as one of his graduate students. His insights and guidance on both manufacturing systems and life in general will prove invaluable after I leave MIT, and I thank him for that. He is a teacher in the purest sense.

I'm also grateful Asbjoern Ambonvik, who provided great insights to this thesis. I also thank Prof. Troxel for allowing me to bring CAFE over to Maxim for use in my thesis, as well as Greg Fischer and Tom Lohman for helping Maxim set CAFE up.

Of course, where would I be without my friends who kept me sane during my time at MIT? They provided the social support structure so dearly needed in a place like this.

Finally, I would like to thank my parents, who have supported me and guided me through the years. There are some debts that can never be repaid.

This work was partially supported by the Advanced Research Projects Agency under contract N00174-93-C-0035.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Objectives and Overview

The objective of this thesis is to create a tool that aids in analyzing an integrated circuit plant's capacity. Although the approach taken towards modeling the manufacturing system and formulating a mathematical problem is similar to other approaches taken before, the approach is unique in the way that the model is simplified and interpreted. Furthermore, this tool, called MUAT (Machine Utilization Analysis Tool), was developed as part of the production analysis tool set available on the CIM (Computer Integrated Manufacturing) system called CAFE (Computer Aided Fabrication Environment).

This thesis touches upon various aspects of a manufacturing system, but focuses on capacity issues. Because manufacturing systems can vary widely from industry to industry, the thesis concentrates on the IC industry to develop a tool best suited to the particular needs of that industry. An optimization model is formulated as the basis from which the tool answers questions regarding capacity. Due to the size and intractability of the model, various techniques are done to simplify and make the model solvable.

## 1.2 Manufacturing and Capacity

Before continuing further, it is first necessary to give an introduction to manufacturing and clarify what capacity means. Because modern large scale manufacturing systems tend to be very complex, understanding their complexity is a prerequisite to appreciating the issues involved in the modeling done later. In addition, the literature on manufacturing is quite large and overall agreement on terminology is lacking. This section also defines some of the terminology used in this thesis.

### 1.2.1 Overview

A manufacturing system is economically justified if it can build products with higher quality or with greater efficiency than other systems can. Specifically, this includes incurring less direct and overhead expenses, replicating physical dimensions as close as possible to design specifications, meeting customer orders on time, etc. To achieve this, the manufacturing system must be tightly controlled. However, control is only possible if the system is well understood.

Unfortunately, many companies run their manufacturing operations without fully understanding them. As a result, quality goes down, too much or too little of a product is produced, and costs increase.

Companies fail to understand their manufacturing systems not because their people are stupid or lazy. Rather, the high stress level and constant "fire fighting" situations focus people on more immediate needs. Understandably, people are more interested in making sure a product gets out the door, rather than take what little time they have to sit down and understand their system.

Furthermore, people often lack tools sophisticated enough to allow them to quickly analyze their manufacturing system. As a result, performing an analysis tends to be time-consuming and difficult to the point that it is just not done.

This thesis describes a tool that allows people to analyze and better understand their manufacturing systems. The tool focuses on one important characteristic: capacity.

### 1.2.2 Capacity Definitions and Discussion

Generally speaking, capacity is the ability to produce. Because everyone operating a manufacturing system needs to know much can be produced, capacity is an important piece of information.

Capacity management is the way the manufacturing system is controlled such that a desired set of products are produced. Skillful management in this area is life or death to a factory. With too much capacity, the factory is severely under-utilized to the point where costs exceed revenues. With insufficient capacity, the factory cannot adequately meet customer needs and the system becomes very slow and inefficient.

Capacity management involves several control factors. Manufacturing resources such as equipment and labor can be augmented, diminished, or distributed around. Customer orders can be restricted or readily accepted. The product mix in the system can be altered. Different scheduling techniques can be implemented.

We now develop a more quantitative definition of capacity. The APICS (American Production and Inventory Control Society) Dictionary [9] defines capacity as "the highest sustainable output rate which can be achieved with the current product specifications, product mix, work force, plant, and equipment".

Unfortunately, defining capacity as an output rate oversimplifies the capacity picture. As Gershwin points out in [8], capacity is too complicated to be defined by a single number but often is misused this way. Gershwin defines an abstract but more precise definition of capacity as "the set of possible production rate vectors, where each component of the vector is the production rate of one of the part types."

In this thesis, capacity is also treated somewhat abstractly. Here, demand exceeds a machine's capacity when the machine is unable to complete all desired operations within a given time frame. At an aggregate level, demand exceeds a manufacturing system's capacity when the system is unable to produce a desired mix and quantity of products within a given time frame.

We now state a well-defined question concerning a manufacturing system's capacity: *Given a finite set of manufacturing resources, is it possible to produce a desired mix and quantity of products within a specified time frame?* From now on, all questions

regarding capacity will be asked in this form.

### 1.2.3 Other Manufacturing System Issues

Capacity is the central issue dealt with in this thesis. However, it is not the only important issue in a manufacturing system. Understanding a manufacturing system requires a breadth of knowledge over many other areas.

One obvious area is cost. How cheaply a manufacturing system can produce its products shows up directly on the company's books. Both average and marginal unit manufacturing costs are closely watched by those in the company responsible for cost control.

Another area that has been caught the attention of the American media during the 1980's has been quality. To build a world class competitive product, manufacturing tolerances must be tight enough to insure that a robust product is always delivered into the customer's hands.

Other areas deal with inventory buffer levels between machines, product lead times, etc. Average inventory levels in buffers between machines are of interest to those who may wish to keep a balanced manufacturing line and low *work-in-process* (WIP) inventory levels. Knowing average product lead time is also necessary to assure customers when their orders will be ready.

Still more areas include production technology, labor relations, etc. These areas are not dealt with here because capacity itself is a very difficult to analyze. However, they are just as important and should also be thoroughly analyzed and understood.

## 1.3 Issues and Literature Review

In the literature on developing models for capacity analysis, there is almost universal agreement that some form of simplification is needed to make the model computationally tractable. The differences in the literature thus primarily stem from the approach taken in modeling as well as how and to what extent simplifications are made.

The simplest model used to analyze capacity are based upon accounting-like meth-

ods such as resource profiling [14] that rely upon a great deal of simplifications and assumptions. These models involve no optimization and are purely deterministic. Many of these models are found in APICS literature such as [14] and [9].

The advantage in such simplistic models lie in their speed of computation. They work reasonable well for very basic manufacturing systems where operation times are constant, machine failures are rare, and no queuing takes place. However, for extremely complicated systems such as those found in the IC industry, these models fail miserably.

Other more sophisticated models are better able to handle such complicated systems but as a price. As the manufacturing systems being analyzed get larger, the effort required to solve these models become quickly prohibitive[1]. These models are based on either stochastic analytical methods, simulation, or optimization techniques.

Most stochastic analytical models are based on Markov chains and queuing theory, a good description and survey of which are found in [8]. Unfortunately, stochastic analytical equations often have limited ability to model complex, real-world phenomena. Such phenomena lie either completely out of what these models can handle, or involve such increased mathematical complexity that they become unsolvable [2]. Of course, many simplifying assumptions and approximations can be made, but to the detriment to the model's accuracy.

Simulation models involve defining relationships between different objects in the manufacturing system, and then running these relationships through time [15]. Simulation models do not require as many assumptions as analytical models do. Instead, they can include all significant factors in the system through brute force calculations. As a result, a wider scope of problems can be addressed. However, a lot of computation time is needed to crunch through the significantly large number of calculations involved in a simulation, making evaluating multiple scenarios for what-if analysis slow. Even for simulation, simplifications need to be made or else the computational effort would be too great. Choosing what simplifications to make is up to the modeler, and is often considered an art.

[1]This phenomenon is sometimes appropriately termed "The Curse of Dimensionality" [3].

The last types of models rely upon optimization techniques. The approach taken by this thesis is based on these techniques, and differs from other approaches taken in the literature primarily in how and to what degree simplifications are made, the resulting stochastic interpretation, and the particular focus on the IC industry. For example, the Lagrangian relaxation techniques taken in [10] are used to decompose large mathematical problems into smaller subproblems which can then be easily solved to generate near optimal schedules. Although [10] is primarily focused on scheduling of a general manufacturing system, the methods used are also applicable in capacity analysis.

In [13], aggregation techniques are the sole methods used to simplify the optimization problem. In addition, [13] concentrates more on developing multi-criteria objective functions for optimization rather than being just concerned with feasibility as is the case in this thesis. [1] also relies upon aggregation to simplify the model, but is instead focused on developing indices calculated from mathematical programs to estimate machine workload.

This thesis also relies upon relaxation and approximation techniques, but of a different form. These techniques, which are fully discussed in Section 3.4, involve relaxing integer restrictions and concentrating on only a subset of the problem's variables and constraints. Relaxing the integer constraints results in a stochastic interpretation not often seen in the literature. Aggregation is also used, but not to the extent carried out in [1] and [13].

## 1.4 Thesis Format

The format of the remaining chapters is as follows. Chapter 2 introduces the IC industry and the complexities particular to IC manufacturing system. Chapter 3 formulates the model used by MUAT in analyzing capacity. This chapter also discusses the difficulties involved in solving the model as well as what simplification techniques are done to help solve it. Chapter 4 illustrates MUAT's use with a toy example. Chapter 5 demonstrates the model's behavior, and discusses the dangers involved

when using the model to analyze manufacturing systems operating near full capacity. Finally, Chapter 6 summarizes this thesis and proposes further research topics.

# Chapter 2

# Integrated Circuit Manufacturing

## 2.1 Description

Manufacturing systems come in many forms. In developing a way to analyze capacity, it is best to pick a particular type of system to concentrate on. This thesis concentrates on IC (Integrated Circuit) manufacturing systems. Analyzing the capacity of these systems presents a challenging problem due to the enormous complexities involved. These complexities are so great that IC companies sometimes do not understand and thus can not fully control their manufacturing systems. To give a better appreciation of the complexity in IC manufacturing systems, background on the nature of the IC fabrication process and industry is now given.

### 2.1.1 Fabrication Process

Figure 2-1 shows the following generalized IC fabrication sequence and its variations.

The fabrication process begins with bare wafers made of silicon or gallium arsenide. These wafers then go through the following sequence of steps:

1a) Material such as silicon dioxide is grown or deposited on the wafer.

and/or

1b) The material properties of the wafers are changed. For example, in an ion implantation process, wafers are bombarded with ions to increase their chemical dopant

concentration.

2) A photolithographic process leaves chemically resistive material called photoresist in some pattern on the surface of the wafers.

3) The wafers are subjected to an etch in a plasma field or chemical wet bath. This etch removes material to a precise depth everywhere except for those areas protected by the photoresist.

4) The photoresist material is then stripped away, leaving a structure mirroring the pattern the photoresist was in.

Note that wafer cleaning, inspecting, and test probing steps are often interspersed between steps.

This sequence is repeated many times, building structure upon structure. Eventually, these structures form solid-state devices with physical feature whose lengths run in the microns. These devices together with interconnecting structures create the circuits which make up the IC. Typically, hundreds of major processing steps and a lead time of six to twelve weeks is needed to complete a finished IC.

To keep track of all the fabrication steps, an IC production sequence follows a process flow, which is a list of all processing steps and their instructions. Figure 2-2 shows a sample portion of such a flow. The Gantt chart in Figure 2-3 shows the typical operation times involved.

Production wafers are typically grouped into *lots* which contain identical wafers going through the same process flow. Wafers in the same lot lot travel together during the fabrication sequence.

### 2.1.2 Industrial Environment

The IC industrial environment can be characterized by its extremely high start-up and production costs. When an IC plant is first built, costs stem mostly from expensive specialized equipment, and clean rooms that only allow a minute number of particles in the air. While the plant is in operation, costs stem mostly from skilled engineers and technicians who command high salaries and wages.

What makes matters worse is that as device size shrinks, and more devices are

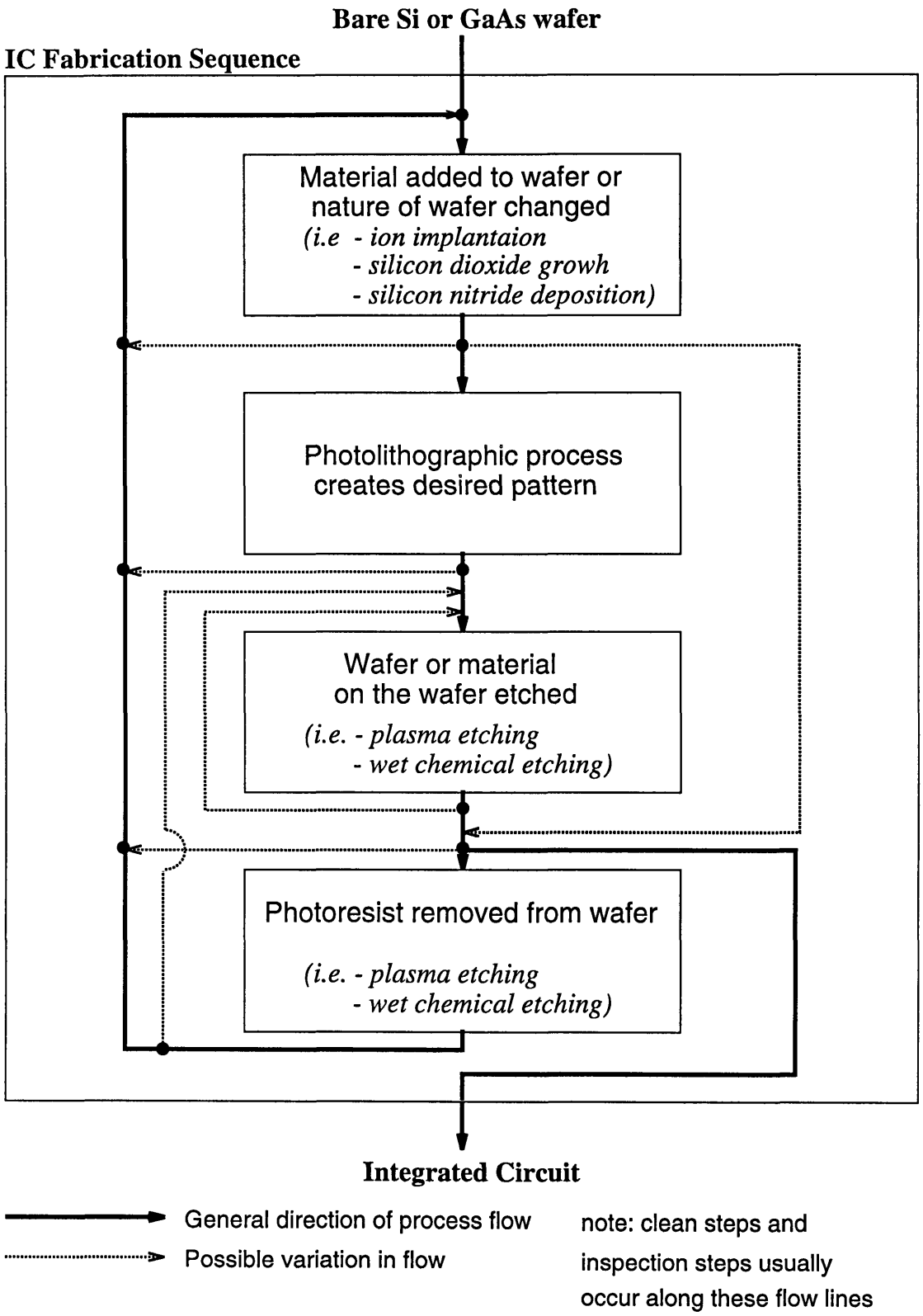


Figure 2-1: IC Process Flow Sequence

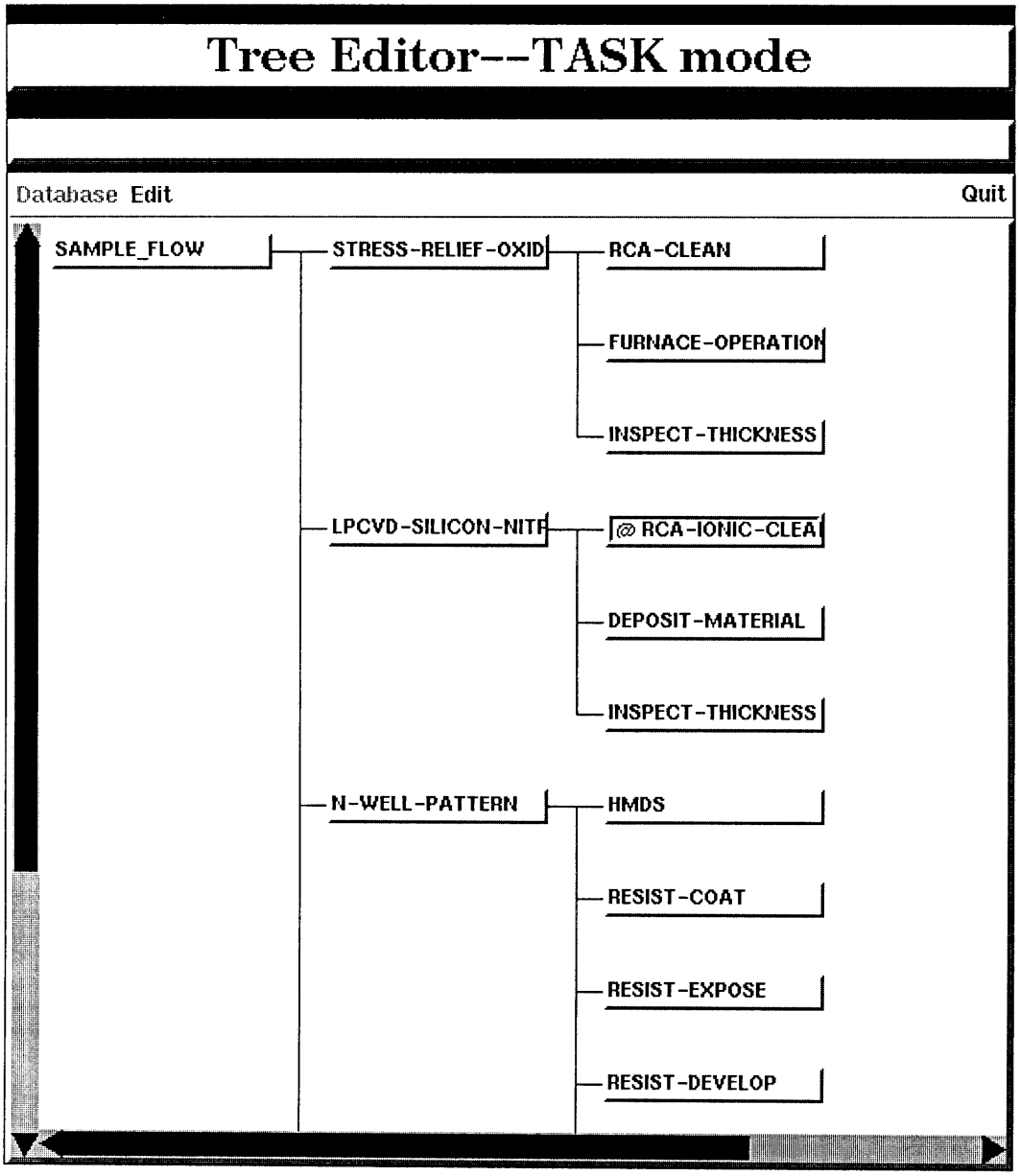


Figure 2-2: Portion of a Process Flow

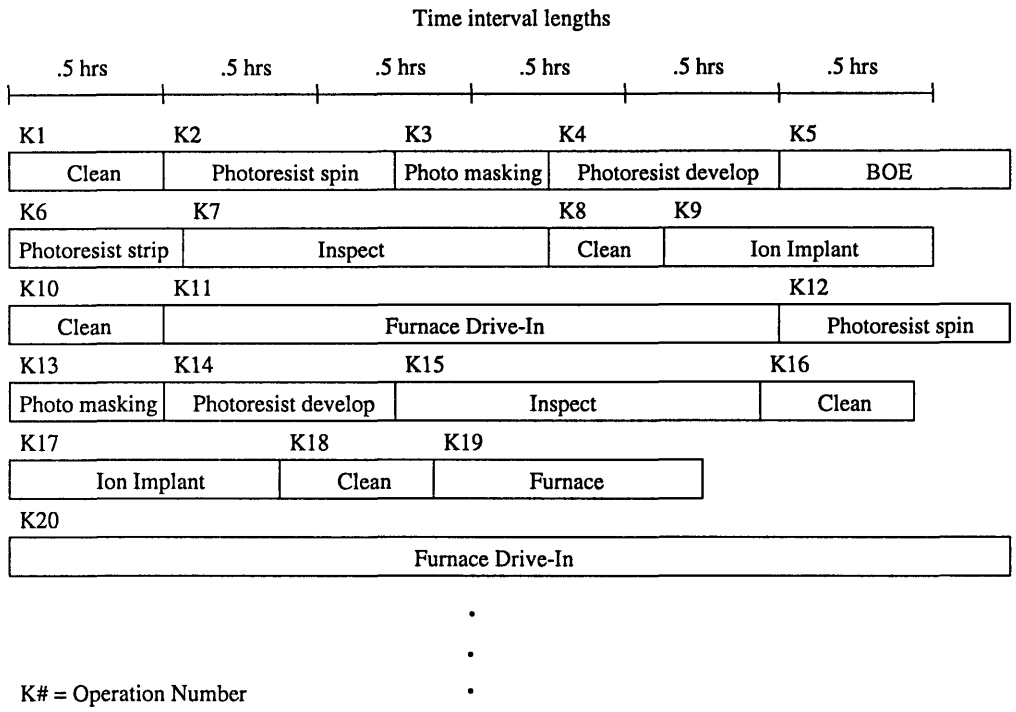


Figure 2-3: Gantt chart of typical IC Process Flow Structure

crammed into a single IC, newer technologies and cleaner environments are needed. Furthermore, production technologies change rapidly so equipment obsolescence is a real danger.

Therefore, it is critical that IC plants quickly financially recuperate from these high capital and operating costs. Products must be produced at high volumes to generate enough revenue while keeping equipment and labor costs minimal. As a result, skilled capacity management of equipment and labor is necessary for success in the IC industry.

## 2.2 Complexities

The complexities of an IC manufacturing system primarily stem from two sources: 1) a high level of unpredictability and 2) a complicated production flow.

The high level of unpredictability is mostly the result of the complex chemical and physical processes involved in IC fabrication which are not fully understood nor in many cases sufficiently modeled. Thus, it is not entirely clear what the results of

a production step will be given a mix of equipment conditions, equipment operator involvement, materials, fabrication environment, and previous fabrication steps. As a result, ad hoc techniques are involved in which a process recipe is used not because it was well understood but because it was found to work through trial and error.

One immediate consequence of a production's step unpredictability is the variability in the amount of time needed to complete that step. Furthermore, wafers may have to be reworked or even scrapped if a step results in a parameter which is out of specification. As a result, product lead time is also unpredictable, especially with queuing effects and machine failures further complicating matters.

Complexity also arises from the complicated IC fabrication sequence. Not only are there hundreds of individual steps, but each product usually goes through a different sequence of steps. Because of this complexity, all IC plants need tracking systems to know where the lots are and what steps should be done next.

An IC production sequence is further complicated by its reentrant nature. Figure 2-1 shows how a lot tends to loop back to certain processes many times in the course of its production sequence. Usually, these processes can only be done on a few particular machines. Thus, these machines tend to be heavily utilized and are often the bottlenecks of the manufacturing system. For example, photolithographic equipment and ion implantation machines tend to be the most heavily utilized equipment in an IC plant.

# Chapter 3

# Problem Statement and Model Formulation

This chapter presents a common problem faced by IC production schedulers and then describes a mathematical model formulated to solve that problem. The main idea behind the model is simplifying and decomposing a large intractable problem into smaller solvable problems. The smaller problems can then be solved using a mathematical programming technique called *linear programming.*

Section 3.1 describes the problem that the tool tries to solve. Section 3.2 gives a brief introduction to mathematical programming. Section 3.3 formulates a mathematical problem that determines the feasibility of a given *release schedule.* Section 3.4 simplifies that problem, and then transforms it into a closely related but more easily solvable problem.

## 3.1 Problem Statement and Motivations

IC production schedulers face a difficult problem when trying to determine the aggregate capacity requirements of their plant's production plans. Their job is to create a list of dates their products must begin processing on so that demand is met on time. These dates are usually referred to as *start dates.* There is no universally accepted term for the list of start dates. Here, it is referred to as a *release schedule.* A sample

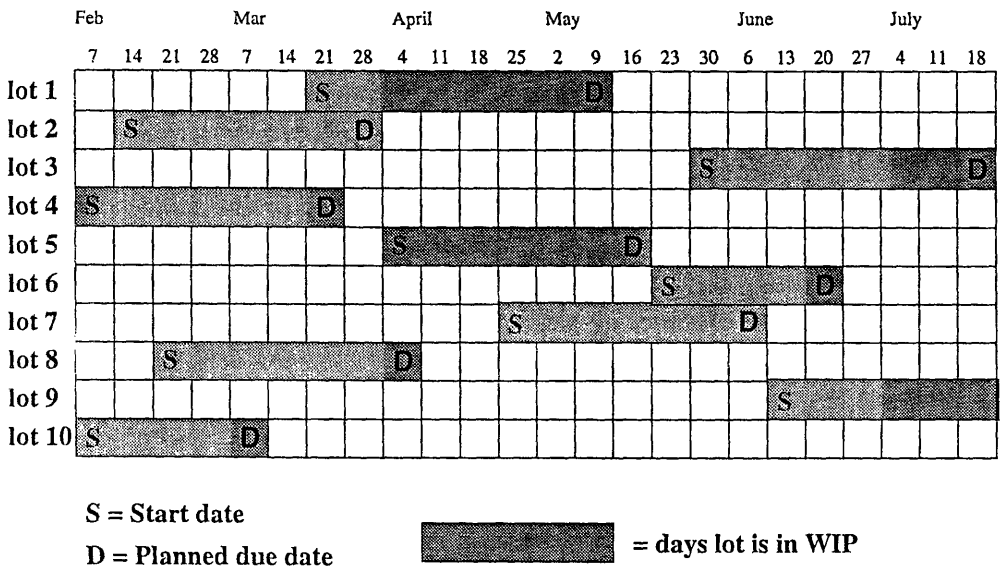


Figure 3-1: Sample Release Schedule

release schedule is given in Figure 3-1. This release schedule specifies a time frame from February 7 until the end of the week of July 18 broken up into periods of equal length, and also specifies the planned dates that lots are due[1]. Note that lots are considered in work-in-process, or *WIP*, from the beginning of the start date's time period to the end of the due date's time period.

The primary problem that this thesis tries to solve is as follows. *Given information about a plant's manufacturing resources and customer demand, is a particular release schedule feasible on average?* A release schedule is *feasible* if lots released on the start dates can be finished by their due dates. A release schedule is *feasible on average* if lots on average finish by their due dates[2]. Note that if machines were down more than usual, then it is possible for a release schedule that is feasible on average, to be infeasible.

The motivations for developing a way to solve this problem lies in the problem's difficulty and importance.

One source of difficulty is the complexity of the IC manufacturing environment

[1]We describe a due date as planned because there is no guarantee that the lot will actually be completed on that date. In fact, lots rarely finish exactly on their due dates.

[2]A more probabilistically correct definition is that a release schedule is feasible on average if the expected times that all lots can finish by are no later than their due dates.

described previously. It is extremely difficult to estimate delivery dates for customers when lead times, yields, and production flow disruptions can be so unpredictable. Furthermore, because a production scheduler may not be able to collect information about process flows, amount of equipment in the plant, etc., the effects of a particular mix of products in a release schedule may not be clear.

The problem's importance arises from the IC industry's high costs. The person responsible for creating release schedules walks a fine line between keeping production costs down and maintaining customer satisfaction. This usually translates into maintaining a minimal set of equipment and personnel while keeping lead times down. Finding the release schedule that achieves this balance is critical if an IC manufacturer wants to stay competitive.

We solve this problem using mathematical programming approaches described next.

## 3.2 Mathematical Programming

### 3.2.1 General Description

An *optimization problem* is a mathematical problem that takes a function of variables called the *objective function* and tries to find the combination of values for these variables that minimizes this function [4]. These variables, called *decision variables* are usually constrained in such a way that they can only take on certain values. The set of all values that these variables can take on is called the *constraint set.* A combination of values that minimizes the objective function is called an *optimal solution.*

Mathematically, an optimization problem can be stated as

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\text{s.t. } \mathbf{x} \in X \end{aligned} \tag{3.1}$$

where $f(\mathbf{x})$ is the objective function, $\mathbf{x} \in \Re^n$ is a vector of $n$ decision variables, and $X$

is the constraint set [4]. The constraint set is comprised of equations called *constraints* which limit the possible values $\mathbf{x}$ can take. Min and s.t. are short–hand for *minimize* and *subject to* respectively.

Mathematical programming refers to the general class of techniques used to solve these optimization problems. These techniques are designed to find an optimal solution as efficiently as possible. Finding an optimal solution often involves two phases. The first phase tries to find a *feasible solution.* A solution is feasible if it lies in the constraint set. If the first phase is successful, the second phase then searches among the feasible solutions until the optimal solution is found.

A mathematical program results in either 1) a single optimal solution, 2) multiple optimal solutions, 3) an infeasible problem, meaning no feasible solutions exist, or 4) an unbounded solution, where one or more decision variables are unbounded.

Mathematical programming problems are usually solved using computer software. The computer package GAMS/MINOS is used to solve the problems formulated in this thesis.

For more information on mathematical programming, please refer to [4] and [12].

### 3.2.2 Linear Programming and Binary Integer Linear Programming

Within mathematical programming, there exist different techniques designed to efficiently solve particular classes of optimization problems having certain structures. Linear programming, or LP, is a set of techniques that are designed for optimization problems with linear objective functions and constraints. A LP problem can be formulated as

$$\begin{aligned} &\min_{\mathbf{x}} \mathbf{c}'\mathbf{x} \\ \text{s.t.} \quad &\mathbf{A}\mathbf{x} \leq \mathbf{b} \end{aligned} \tag{3.2}$$

where $\mathbf{x} \in \Re^n$ is a vector of decision variables, $\mathbf{c} \in \Re^n$ is a vector of cost coefficients, and $\mathbf{A}$ and $\mathbf{b}$ are a matrix and vector respectively of coefficients such that $\mathbf{A}\mathbf{x} \leq \mathbf{b}$

form linear inequalities [12].

Another set of techniques related to linear programming is *mixed integer linear programming*, or MILP. MILP problems are LP problems where some decision variables can only be integer valued, while other variables can take on the entire set of real numbers. Here, we deal with a subset of MILP called *binary integer linear programming*, or binary ILP. In binary ILP problems, all decision variables can only equal 1 or 0 [12].

Although LP and binary ILP problems differ only by this added constraint, their solution methods differ greatly. The most noticeable difference is the time it takes to solve their respective problems. Linear programs are usually solved by using simplex or interior point methods [15]. It is beyond the scope of this thesis to discuss the details of these methods, but suffice it to say that these methods can usually find optimal solutions relatively quickly.

On the other hand, methods for solving binary ILP problems are very slow. The reason is that once the 0-1 constraint is added to a LP problem, the problem becomes *combinatorial* [12]. Combinatorial problems are optimization problems in which the decision variables can take only discrete values and the constraint set consists of a finite number of combinations of these discrete values. Again, it is beyond the scope of this thesis to explain in detail why, but the time to solve combinatorial problems often increases exponentially with problem size [12]. Therefore, if a binary ILP and LP problem were of similar size, the LP problem could be solved more quickly. In fact, for very large binary ILP problems, it would take literally years to find an optimal solution.

## 3.3 Initial Formulation

In this section, we formulate a binary ILP problem to mathematically model the problem stated in Section 3.1: Given information about a plant's manufacturing resources and customer demand, is a particular release schedule feasible on average?

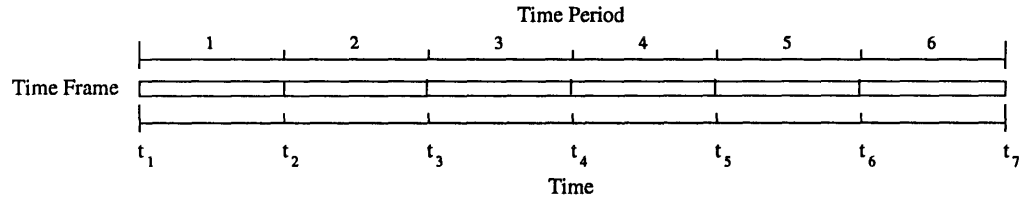


Figure 3-2: Distinction Between Time Period and Time

### 3.3.1 Indexes and Definitions

Let there be $I$ equal time periods in the release schedule's time frame, and let time period $i$=$\{1, \ldots, I\}$ be the $i$'th time period in that time frame. Let $P$ be the length of a time period, which we assume to be greater than the length of the longest operation time.

Let $t_i$ be the time that separates time period $i-1$ from time period $i$. Time period $i$ thus refers to the interval of time between $t_i$ and $t_{i+1}$. This distinction between time period and time is clarified in Figure 3-2.

Let there be $L$ lots in the system during the time spanned by the time frame, and let lot $l$=$\{1, \ldots, L\}$ be the $l$'th lot. Let operation $k$=$\{1, \ldots, K_l\}$ be the $k$'th operation in a lot's process flow, where $K_l$ is the total number of operations in lot $l$'s process flow. For the sake of simplicity, an operation will be described as just belonging to a lot rather than to the lot's process flow.

Let machine type $m$ be the $m$'th *machine type.* A machine type is a group of identical machines that are interchangeable with each other. Define $E_m$ as the number of machines in the $m$'th machine type.

An operation to be done on a lot is referenced by an ordered pair $(k, l)$. Given an operation $(k, l)$, a machine type $m$ is determined. Note that many different pairs of $(k, l)$ correspond to the same $m$.

This relationship between $(k, l)$ and $m$ can be described using sets. Let $M$ be the set of all $(k, l)$ corresponding to operations. Let $M_m$ be the set of all $(k, l)$ that determine machine type $m$. Note that $M_m$ are non–intersecting sets, and $M$ is the union of all $M_m$.

Let $S_l$ be the time period whose beginning corresponds to lot $l$'s start date. Pro-

cessing on that lot can start at the beginning of that time period. Let $D_l$ be the time period that lot $l$ is due. Assume lot $l$ must leave WIP sometime during time period $D_l$. Note that $t_{S_l}$ and $t_{D_l+1}$ are the times corresponding to the start of time period $S_l$ and end of time period $D_l$ respectively. We also assume that $t_{D_l+1} - t_{S_l} + P$ is not less than the sum of processing times in the lot's process flow.

Let $w_l^k(i)$ be the decision variable in the Binary ILP problem. $w_l^k(i)$ is defined as

$$w_l^k(i) = \begin{cases} 1 & \text{if lot } l\text{'s operation } k \text{ occurs during time period } i \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Let $U_l^k(i)$ be defined as

$$U_l^k(i) = \begin{cases} 1 & \text{if lot } l\text{'s operation } k \text{ has occurred by the end of time period } i \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

$U_l^k(i)$ can thus be expressed as

$$U_l^k(i) = \sum_{j=S_l}^{i} w_l^k(j) \quad \forall \; k, l, S_l \leq i \leq D_l - 1 \tag{3.5}$$

Note that because $U_l^k(D_l)$ is always equal to 1 for $(k, l) \in M$ if the problem is feasible, the case of $i = D_l$ is not included in Equation (3.5).

Let $\tau_l^k$ be the length of time in hours that lot $l$'s $k$'th operation spends on a machine[3]. Therefore,

$$\sum_{\substack{kl \\ (k,l) \in M_m}} \tau_l^k w_l^k(i)$$

represents the total load in hours on machine type $m$ during time period $i$.

Let $N$ be the number of work hours in a time period. Assume each time period has the same number of work hours. Let $e_m$ be the average fraction of time that machine type $m$ is available after machine failures and preventive maintenance. $e_m$ is commonly known as the *efficiency* of the machine type. Since one machine can at

[3]As mentioned before, this processing time is often uncertain. As a result, $\tau_l^k$ is usually either estimated or given a worst case value.

most be occupied for $Ne_m$ hours each time period, the total number of hours actually available in a time period for lots to be processed on machine type $m$ is $Ne_m\ E_m$.

However, keeping machines running 100% of the time is unwanted since the manufacturing system becomes unstable, resulting in high WIP inventory levels and long lead times [8]. Let $\xi$ be the fraction of available time the user desires the machine to be busy. A $\xi$ of .8 represents a desire to run machines only 80% of the time they are available. Therefore, $Ne_m \xi E_m$ is the limit of how many hours in a time period machine type $m$ can be working.

### 3.3.2 Constraints

The first constraint is the 0-1 restriction that makes the formulation a binary ILP problem. This is written as

$$w_l^k(i) \in (0,1) \quad \forall \ k, l, S_l \leq i \leq D_l \tag{3.6}$$

The next two constraints model $w_l^k(i)$ as defined in Equation (3.3). Together with Equation (3.6), the first of these constraints

$$\sum_{i=S_l}^{D_l} w_l^k(i) = 1 \quad \forall \ (k,l) \in M \tag{3.7}$$

states that a lot's operation must occur, and that it may occur in only one time period. The second of these constraints

$$w_l^k(i) = 0 \quad \forall \ (k,l) \notin M \tag{3.8}$$

ensures that $w_l^k(i)$ is zero for those $(k,l)$ pairs that are not meaningful.

The next constraint that the decision variables $w_l^k(i)$ are subject to is the limited amount of time a machine type can be used in a time period. This capacity constraint can be written as

$$\sum_{\substack{kl \\ (k,l)\in M_m}} \tau_l^k w_l^k(i) \leq Ne_m \xi E_m \quad \forall \ m, i \tag{3.9}$$

This equation states that for each time period and machine type, the total load on that machine type must not be greater than the total number of hours the machine type can be working.

Another constraint in the formulation is needed to deal with operation *precedence.* Precedence means than operations must occur in the order that they are listed in their process flow. This can be represented as

$$U_l^{k+1}(i) \leq U_l^k(i) \quad \forall \; k, l, S_l \leq i \leq D_l - 1 \tag{3.10}$$

In other words, a lot's operation can not take place before the time period during which the previous operation took place. Note that we are implicitly ignoring the situation where $D_l = S_l$, in which case Equation (3.10) is automatically satisfied.

The last constraint takes into account the limited number of hours a lot can be processed on in a time period.

$$\sum_{\substack{k \\ (k,l) \in M}} \tau_l^k w_l^k(i) \leq N \quad \forall \; l, S_l \leq i \leq D_l \tag{3.11}$$

This says that the sum of process times of operations performed on a lot during some time period may not exceed the number of hours during a time period.

Equation (3.9) may not be adequate when machine loads are close to maximum capacities because queuing effects are ignored in the model. Queuing effects arise from the stochastic nature of lot arrivals at a machine as well as from variable processing times. These effects are not taken into account because they are quite difficult to model. However, queuing problems generally only become severe when the system is operating near full capacity.

Fortunately, ignoring queuing effects does not turn out to be too big of a problem for the following reasons. First of all, using an appropriate $\xi$ should keep the model from experiencing machine loads close to maximum capacities. Secondly, even if machines were operating at maximum capacities, as will be explained in later when the problem is simplified, the precision of the solutions is already questionable.

### 3.3.3 Solution Considerations and Difficulties

The binary ILP problem formulation is summarized as

$$\min \sum_{l=1}^{L}\sum_{k=1}^{K}\sum_{i=S_l}^{D_l} c_l^k(i) w_l^k(i) \tag{3.12}$$

s.t.

$$\begin{array}{rcll}
\sum_{i=S_l}^{D_l} w_l^k(i) & = & 1 & \forall\ (k,l) \in M \\
w_l^k(i) & = & 0 & \forall\ (k,l) \notin M \\
U_l^k(i) & = & \sum_{j=S_l}^{i} w_l^k(j) & \forall\ k,l,S_l \leq i \leq D_l - 1 \\
\sum_{\substack{kl \\ (k,l)\in M_m}} \tau_l^k w_l^k(i) & \leq & N e_m \xi E_m & \forall\ m,i \\
U_l^k(i) & \leq & U_l^k(i) & \forall\ k,l,S_l \leq i \leq D_l - 1 \\
\sum_{\substack{k \\ (k,l)\in M}} \tau_l^k w_l^k(i) & \leq & N & \forall\ l,S_l \leq i \leq D_l \\
w_l^k(i) & \in & (0,1) & \forall\ k,l,S_l \leq i \leq D_l
\end{array}$$

where $c_l^k(i)$ are the cost coefficients in the objective function.

As mentioned in Section 3.2, optimization sometimes involves two phases, the first being concerned with feasibility, and the second with optimality. The first phase only looks at the constraints, while the second phase looks at both the constraints and the objective function. Since this thesis is only concerned with feasibility, the objective function does not matter as long as it is linear in $w_l^k(i)$. Hence, the choice of the cost coefficients $c_l^k(i)$ above does not really matter. It may be useful to choose $c_l^k(i)$ so that the second phase is gone through faster or so that the model's output can be used as an input to a production scheduling problem. However, these two concerns are beyond the scope of this thesis.

Although this formulation captures all the necessary constraints from which to determine the feasibility of a release schedule, the following difficulties exist. First of all, the time it takes to solve a binary ILP problem increases exponentially with the number of variables. A typical release schedule may have hundreds of lots, each having hundreds of operations, which could take place in one of many different time periods. Therefore, solving such a problem could take on the order of years, and no existing binary ILP solver can handle such a large problem.

To remedy this difficulty, certain problem reduction techniques are carried out to transform this intractable binary ILP problem into a smaller, easily solvable LP problem. All this is the topic of the next section.

## 3.4 Problem Reduction Techniques

Three techniques, aggregation, selected focus, and relaxation, are used to make the problem solvable. These techniques involve making approximations and assumptions to greatly simplify the problem. Approximations and assumptions are central to creating solvable models of reality. They allow us to obtain solutions sufficiently close to the actual solutions of larger, unsolvable problems. The first two techniques reduce the number of variables needed. Aggregation groups lots together and divides the time frame into larger time periods. Selected focus reduces the original problem into a smaller one with fewer machine types. The last technique, relaxation, changes the structure of the problem into a LP formulation.

### 3.4.1 Aggregation

Aggregation is the easiest technique used to simplify the problem. The user of the tool can combine into one, those lots in the release schedule having identical or close to identical process flows, start, and due dates. The user can also discretize over larger time units in the release schedule. Note that how much is aggregated depends on the user. The tool does not automatically aggregate variables to simplify the problem.

Aggregation is a trade-off between faster solutions and a less granular model of reality. The more aggregation that occurs, the less precise the model's output is. For example, a model with too large a time period would ignore capacity problems that could occur on much smaller time scales.

### 3.4.2 Selected Focus

Selected focusing is based on the premise that there exist certain *critical machine types* in the production process. A machine type is critical if it tends to be heavily utilized. Critical machines are also commonly known as *bottlenecks*. The problem can then be reduced to one that only worries about exceeding the capacity of these critical machine types. All other machine types are assumed to always have enough capacity to process their operations. Therefore, it is assumed that an operation using a non-critical machine type is started as soon as the previous operation in the process flow is finished. Operations using critical, and non-critical machine types will be termed *critical* and *non-critical operations*, respectively.

Let $M_m \in \widetilde{M}$ for all $m$ such that machine type $m$ is a critical machine type. Hence, $\widetilde{M}$ is the subset of $M$ of critical machine types.

As a result, Equations (3.7) and (3.8) become

$$\sum_{i=S_l}^{D_l} w_l^k(i) = 1 \quad \forall \ (k,l) \in \widetilde{M} \tag{3.13}$$

$$w_l^k(i) = 0 \quad \forall \ (k,l) \notin \widetilde{M} \tag{3.14}$$

Equations (3.9), (3.10), and (3.11) also change to

$$\sum_{\substack{kl \\ (k,l)\in M_m}} \tau_l^k w_l^k(i) \leq N e_m \xi E_m \quad \forall \ i, \left\{ m \mid M_m \in \widetilde{M} \right\} \tag{3.15}$$

$$U_l^{\bar{k}}(i) \leq U_l^k(i) \quad \forall \ k, l, S_l \leq i \leq D_l - 1 \tag{3.16}$$

$$\sum_{\substack{k \\ (k,l)\in \widetilde{M}}} \tau_l^k w_l^k(i) \leq N \quad \forall \ l, S_l \leq i \leq D_l \tag{3.17}$$

where $\bar{k}$ is the next critical operation after operation $k$, assuming such an operation exists. Note that Equation (3.17) is now a weaker statement because non-critical

machine operations are not taken into account. This is an unfortunate, but necessary price to pay for being able to solve a smaller problem.

To use this technique, it is first necessary to know which machine types are critical. However, this information is often not known a priori since it is dependent on product mix as well as start and due dates.

Fortunately, users of the tool can usually make a good initial guess as to which machine types are critical. The solution of the problem becomes an iterative process as follows. Given an initial list of critical machines types the tool evaluates a release schedule. If the release schedule is infeasible, then the initial guess was adequate in allowing the detection of capacity problems.

If the release schedule is feasible, a solution containing a feasible set of $w_l^k(i)$ is returned. However, because Equation (3.15) only checked critical machines for feasibility, it is possible that Equation (3.9) would be violated for other machines erroneously thought to be non-critical. This violation can be checked for by interpolating when non-critical operations occur, and then estimating the load on each non-critical machine type for each time period.

If the load on a non-critical machine type is greater than the number of hours the machine type can be working, then the critical machine types had been guessed wrong. The non-critical machines types that violated Equation (3.15) should then be added to the critical machine list. If the loads on machine types previously thought to be critical are low, those machine types should then be removed from the list.

The process is repeated until either an infeasible solution occurs or a feasible solution occurs for both critical and non-critical machine types. The tool automatically assumes that selected focusing is done, but it is up to the user to create and update the critical machine list after each iteration.

Note that selected focusing does not work if the line is balanced and no bottlenecked machines exist for a given release schedule. In that case, no problem exists since no bottleneck machines implies all machines have ample capacity. Therefore, the release schedule must be feasible.

### 3.4.3 Relaxation

Despite the vast reduction of variables in the model using the aggregation and selected focus techniques above, the problem is still too large for binary ILP algorithms to solve in finite time. Most binary ILP solvers are limited to only a hundred or so variables. For a realistic system, the model requires tens of thousands of variables, thus lying far from the capabilities of any ILP solvers today.

The last technique, relaxation, reduces the original problem by converting it from a binary ILP formulation into a LP. Relaxation is a common technique used to solve difficult ILP problems, and generally refers to ignoring the integer restriction [12].

For the binary ILP problem above, the binary constraint

$$w_l^k(i) \in (0,1) \quad \forall \; k, l, S_l \leq i \leq D_l$$

relaxes to

$$0 \leq w_l^k(i) \leq 1 \quad \forall \; k, l, S_l \leq i \leq D_l \tag{3.18}$$

Using relaxation techniques to solve an ILP problem involves building an integer solution from the relaxed problem's solution. However, building an integer solution can be quite involved and difficult. Instead, we keep the relaxed solution as is, and interpret $w_l^k(i)$ a little differently.

Before, $w_l^k(i)$ was a variable defined as

$$w_l^k(i) = \begin{cases} 1 & \text{if lot } l\text{'s operation } k \text{ occurs during time period } i \\ 0 & \text{otherwise} \end{cases}$$

An operation either occurred during a certain time period, or it did not.

Under the new relaxed problem, $w_l^k(i)$ can be interpreted as a probability. Generally speaking, probability is the quantification of ignorance [7]. By stating the probability of an event occurring in some situation, we admit that we do not know with certainty whether or not the event will occur in a single execution of that situation. Instead, we are saying that if that situation was repeated many times, the

long-run average frequency of that event occurring would be equal to the value of that probability. For example, if a coin was tossed many times under the same set of conditions, the long-run average frequency that a head appears per toss would be equal to one-half. Under this interpretation, $w_l^k(i)$ is now the long-run average frequency that lot $l$'s $k$'th operation occurs during time period $i$ if the manufacturing system were to use the same release schedule many times.

Since we still have

$$\sum_{i=S_l}^{D_l} w_l^k(i) = 1 \quad \forall \ (k,l) \in \widetilde{M}$$

and

$$w_l^k(i) = 0 \quad \forall \ (k,l) \notin \widetilde{M},$$

$w_l^k(i)$ can be interpreted as the probability that lot $l$'s $k$'th operation occurs during time period $i$. Note that because Equation (3.13) implies

$$\sum_{i=S_l}^{D_l} w_l^k(i) \leq 1 \quad \forall \ (k,l) \in \widetilde{M},$$

Equation (3.18) reduces to

$$w_l^k(i) \geq 0 \quad \forall \ k, l, S_l \leq i \leq D_l \tag{3.19}$$

Under the above interpretation of probability,

$$\sum_{\substack{kl \\ (k,l) \in M_m}} \tau_l^k w_l^k(i)$$

in now the long-run average load on machine type $m$ during time period $i$ if the manufacturing system were to use the same release schedule many times. Therefore, Equation (3.15) now states that for each time period and critical machine type, that long-run average load must not be greater than the total number of hours the machine type can be working. Also, $U_l^k(i)$ can now be interpreted as the probability that lot $l$'s $k$'th operation has occurred by $t_{i+1}$. Equation (3.16) states that for a particular

lot and time period, the probability that the $k$'th operation has occurred must not be less than the probability that the next operation using a critical machine type has occurred. Finally, Equation (3.11) is now interpreted as stating that the average sum of process times of critical operations performed on a lot during some time period may not exceed the number of hours during that time period.

It is convenient to define the variable $V_l^k$ as the expected time that the $k$'th operation of lot $l$ will occur. We assume that operations begin as early during a time period as possible. Therefore, we can define $V_l^k$ as

$$V_l^k = \sum_{i=S_l}^{D_l} w_l^k(i) t_i \quad \forall \ k, l \tag{3.20}$$

For example, $w_l^k(i) = 1$, then $V_l^k = t_i$. If $w_l^k(i) = .5$ and $w_l^k(i+1) = .5$, then $V_l^k = .5t_i + .5t_{i+1}$, and so on.

Using this definition of $V_l^k$, we can state another constraint concerning the precedence of operations as follows.

$$V_l^{\bar{k}} \geq V_l^k + T_l^k \quad \forall \ (k, l) \in \widetilde{M} \tag{3.21}$$

where $\bar{k}$ is the next critical operation after operation $k$, assuming such an operation exists, and $T_l^k$ is the sum of the process times of all operations after and including that of the $k$'th operation up to the $\bar{k}$'th operation. Equation (3.21) states that the expected time a critical operation occurs must not be less than the expected time the previous critical operation occurs plus all the processing time between them.

Because non-critical operations can be at the very beginning or end of a process flow, the following two additional inequalities are needed:

$$V_l^{beg} \geq t_{S_l} + T_l^{beg} \quad \forall \ l \tag{3.22}$$

where $V_l^{beg}$ is the first critical operation in lot $l$'s process flow, and $T_l^{beg}$ is the sum of the process times of all non-critical operations before that first critical operation,

and

$$V_l^{end} \leq t_{D_l} + P - T_l^{end} \quad \forall \ l \tag{3.23}$$

where $V_l^{end}$ is the last critical operation in lot $l$'s process flow, and $T_l^{end}$ is the sum of the process times of all operations after and including the last critical operation. Equation (3.22) states that the expected time the first critical operation occurs must not be less than the beginning of time period $S_l$ plus all the processing time before that operation. This must be true because we assume that lots are released at the beginning of time period $S_l$. Equation (3.23) states that the expected time the last critical operation occurs must not be greater than the end of time period $D_l$ minus all the processing time after the start of that operation. This must be true because we assume that lots are due sometime during time period $D_l$.

Note that since $V_l^k$ are the times when critical operations are expected to occur, it is possible to use these $V_l^k$'s to interpolate when non-critical operations are expected to occur. This interpolation is used to estimate the load on non-critical machines types for the purposes described in Section 3.4.2.

### 3.4.4 Final Formulation Summary

After all the problem reductions techniques have been done, the problem can be summarized as follows

$$\min_{ilk} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{i=S_l}^{D_l} c_l^k(i) w_l^k(i) \tag{3.24}$$

s.t.

$$\begin{aligned}
\textstyle\sum_{i=S_l}^{D_l} w_l^k(i) &= 1 && \forall\ (k,l) \in \widetilde{M} \\
U_l^k(i) &= \textstyle\sum_{j=S_l}^{i} w_l^k(j) && \forall\ k,l,S_l \leq i \leq D_l - 1 \\
V_l^k &= \textstyle\sum_{i=S_l}^{D_l} w_l^k(i) i && \forall\ k,l \\
\textstyle\sum_{\substack{kl \\ (k,l)\in M_m}} \tau_l^k w_l^k(i) &\leq N e_m \xi E_m && \forall\ i, \left\{ m \mid M_m \in \widetilde{M} \right\} \\
U_l^{\bar{k}}(i) &\leq U_l^k(i) && \forall\ k,l,S_l \leq i \leq D_l - 1 \\
\textstyle\sum_{\substack{k \\ (k,l)\in M}} \tau_l^k w_l^k(i) &\leq N && \forall\ l, S_l \leq i \leq D_l \\
V_l^{\bar{k}} &\geq V_l^k + T_l^k && \forall\ (k,l) \in \widetilde{M} \\
V_l^{end} &\leq D_l - T_l^{end} && \forall\ l \\
V_l^{beg} &\geq S_l + T_l^{beg} && \forall\ l \\
w_l^k(i) &\geq 0 && \forall\ k,l,S_l \leq i \leq D_l
\end{aligned}$$

## 3.5 Model Solution and Output

This section explains how the model is solved and how its results are interpreted. First, a description is given on the solution method for this model. Next, the meaning of the probabilities and expected values returned by the model are explained. Then, some guidelines are given on how to determine the feasibility of a release schedule from the model's results.

### 3.5.1 Solution

As mentioned in Section 3.2.1, mathematical programs can be solved using the software package GAMS/MINOS. GAMS (General Algebraic Modeling System) is a front-end language by which mathematical programs can be easily formulated and coded in [6]. MINOS is a program that is then called to efficiently solve that mathematical program. Because the final formulation above is a linear program, MINOS is able to use the simplex method as its solution algorithm.

The solution returned by the GAMS/MINOS package is the set of $w_l^k(i)$ that minimizes the objective function above. The values of the variables returned by GAMS are thus influenced by the choice of the objective function. Although this choice does not affect the feasibility of the problem, it affects which feasible solution is selected as the optimal one returned. Therefore, unless a cost function was chosen with a particular purpose in mind, it does not matter what the specific values of $w_l^k(i)$, $V_l^k$, and the average machine loads are. What matters more are general observations such as how machine types tend to be loaded. A more detailed interpretation of the solution returned by GAMS/MINOS are given next.

### 3.5.2 Interpretation

We have already given a stochastic interpretation of $w_l^k(i)$, $U_l^k(i)$, and $V_l^k$ above. To summarize, $w_l^k(i)$ is the long-run average frequency that lot $l$'s $k$'th operation occurs during time period $i$ if the manufacturing system were to use the same release schedule many times. $U_l^k(i)$ is the probability that lot $l$'s $k$'th operation has occurred by $t_{i+1}$. $V_l^k$ is the time that lot $l$'s $k$'th operation is expected to occur on average.

It is important to remember that $V_l^k$ must not be interpreted as the precise time when an operation should occur. The values of $V_l^k$ returned by the model should not be treated as a schedule for operation start times.

It should also be noted that the model does not assume any feedback from reality. If the release schedule was actually carried out, events such as operations being performed and machine disruptions are not explicitly fed back into the model. Instead, the model takes into account the aggregate long-run effects of all these events by including their long-run frequencies in the calculations. For example, although certain machine failures and repairs may take place while the manufacturing system is running, the model has already taken them into account by including the long-run average efficiencies of these machines.

Translating the values of the variables returned by GAMS into a closed loop schedule involves many considerations not treated here. It is an important research problem that should be studied, but whose solution is far from obvious.

### 3.5.3 Release Schedule Feasibility

Inputting a release schedule into the model and solving the problem with GAMS results in a unique optimal solution, multiple optimal solutions, an unbounded solution, or an infeasible solution, as described in Section 3.2.1. If the solution is infeasible, then the manufacturing system's limited physical resources cannot handle the time constraints imposed by the release schedule and the process flows. Fortunately GAMS still returns the set of values for $w_l^k(i)$ and $V_l^k$ that satisfy as many constraints as possible. The user can then use this information to determine which machines types have loads which are close to or have exceeded maximum capacity.

If GAMS finds that the problem is feasible and returns an optimal solution, the user must use the Assumption Checker Module to first make sure that none of the critical machine assumptions were violated as was described in Section 3.4.2. If the assumptions were violated, then the list of critical machines must be updated and the problem rerun. However, if none of the critical machine assumptions were violated, then the release schedule is feasible according to the model.

Unfortunately, if GAMS says that the release schedule is feasible. this does not necessarily mean that the release schedule is feasible in reality. This uncertainty about feasibility is the result of the trade-off made when the original binary ILP problem was relaxed. By relaxing the binary ILP problem, a solvable LP problem was gained. But since the two problems are not totally equivalent, a feasible solution in the LP problem is quasi-feasible in the binary ILP problem. Quasi-feasible means that a release schedule that is feasible for the binary ILP problem is always feasible for the LP problem. However, if the LP problem finds a feasible solution for some release schedule, it does not necessarily mean that the binary ILP problem will be feasible for that same release schedule.

The reason is that the results of the LP problem are feasible long-run average values if the release schedule was inputted into the same system many times. The model says nothing about the variability around these averages. Therefore, although average machine loads may be feasible, maximum capacity might actually be exceeded in any particular execution of the release schedule. The closer the system is to full

capacity, the greater the probability that a binary ILP problem would be infeasible for a release schedule that the LP problem is able to find a feasible solution for. As a result, if the model returns average loads on machines that are a little less than their maximum capacities, it is likely that the release schedule will be not be feasible in reality.

Chapter 4 gives an example of the tool's use and illustrates how to determine a release schedule's feasibility from the tool's results.

# Chapter 4

# MUAT Description and Example of Use

## 4.1 MUAT Overview

### 4.1.1 Purpose

The purpose of MUAT (Machine Utilization Analysis Tool) is to provide an easy way for users to quickly collect information for and solve the model formulated in Chapter 3. MUAT also provides an easy-to-understand graphical representation of the model's results. By automating these tasks, MUAT can be used by production schedulers to quickly try out different release schedules. Thus, MUAT can be a heuristic aid that allows production schedulers to get a sense of the work load imposed on machines as well as potential capacity problems resulting from each schedule. At the same time, production schedulers also gain more understanding of how various factors affect the capacity of their manufacturing system.

### 4.1.2 External Programs Used

MUAT utilizes three external packages called CAFE, GAMS/MINOS, and XFIG to accomplish its purpose. In a way, MUAT can be thought as a set of interfaces between these external packages, which are now described.

CAFE (Computer Aided Fabrication Environment) is a CIM (Computer Integrated Manufacturing) program that has been developed by the CIDM (Computer Integrated Design and Manufacturing) group at MIT to support IC fabrication [11]. CAFE has information about process flows of all products in the fabrication environment CAFE supports. CAFE allows other programs to use this process flow information.

GAMS/MINOS is a program that allows a mathematical problem to be compactly formulated and solved. GAMS reads in a text file that contains the mathematical problem coded in the GAMS modeling language, solves the problem with a user-specified algorithm, and returns the problem's solution in another text file.

The XFIG program [16] is the primary way that the tool's graphical results are displayed. XFIG reads in a text file containing drawing objects, and then graphically displays these objects. XFIG also has the ability to translate the drawing into such formats as Postscript and Latex.

## 4.2 Description of MUAT Modules and Example of Use

A description of the tool's modules and their interfaces is now presented. An example of MUAT's use is also concurrently given. For the sake of tractability, an example with a release schedule containing hundreds of lots having process flows as complicated as that shown in Figure 2-3 is not given. Rather, a much scaled down toy example is gone through. The process flows used in this toy example captures some of the characteristic traits of IC process flows, namely their re-entrant natures.

MUAT is written in C and is divided into 4 modules: the Preprocessor, Matrix Generator, Analysis, and Assumption Checker Module. Figure 4-1 gives a clearer picture of the tool's modules, and how they interface with CAFE, GAMS, and XFIG.

The Preprocessor module's purpose is to alleviate the computational burden of other modules having to directly extract data from CAFE. The Preprocessor module caches process flow information from CAFE into text files from which data can be

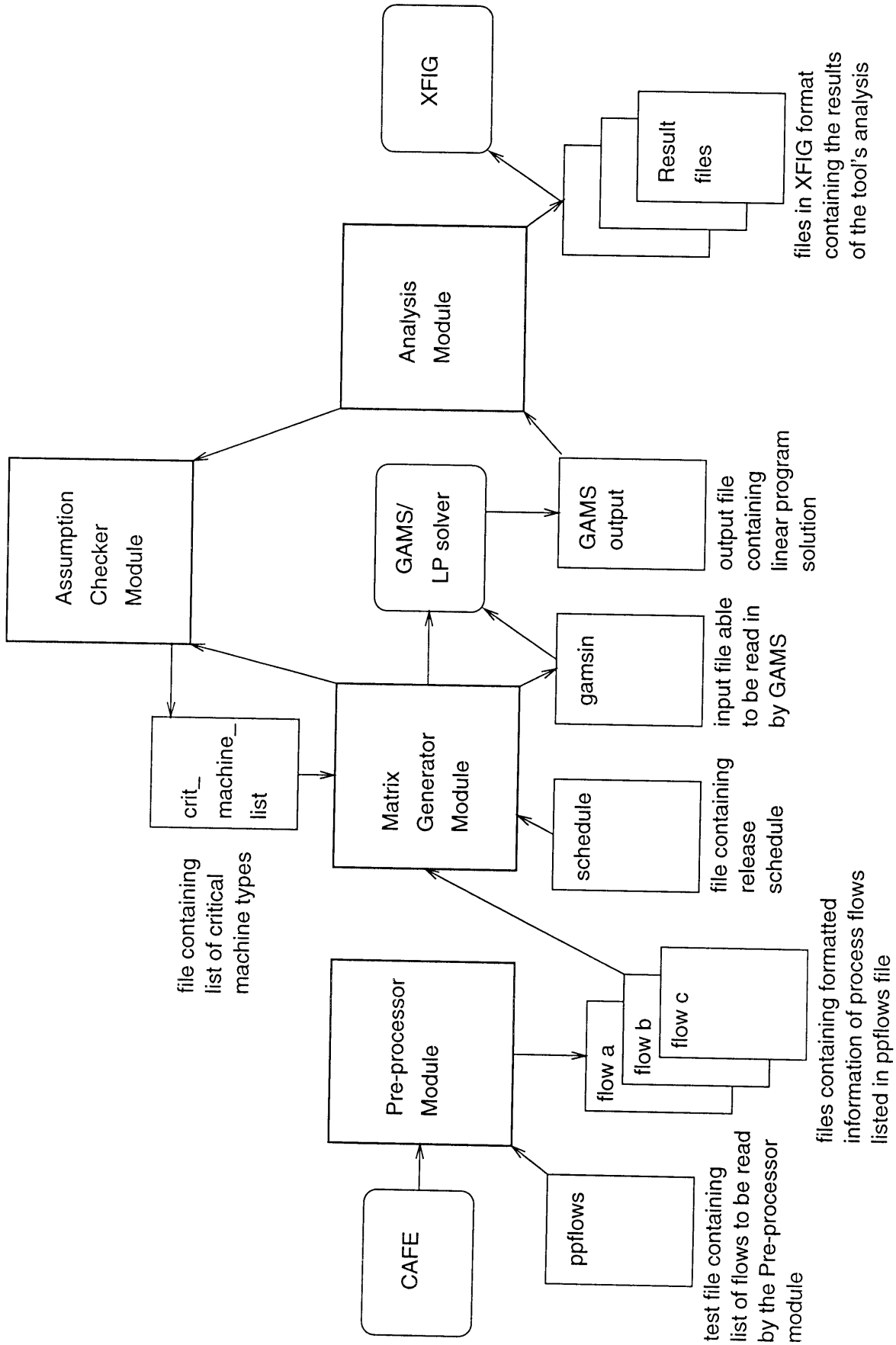


Figure 4-1: Capacity Tool's System Design

pfA
pfB

Figure 4-2: *ppflows* file containing list of process flows names

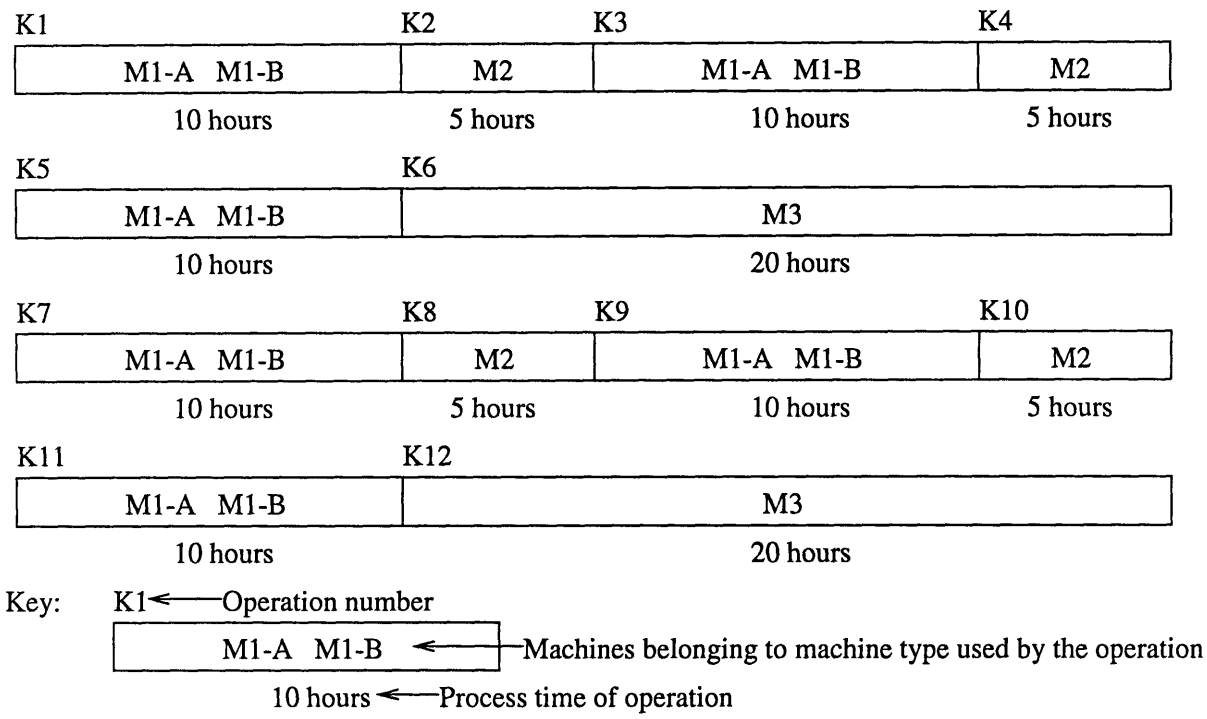


Figure 4-3: Gantt chart for process flow pfA

extracted faster than if data was extracted from CAFE itself.

The module reads in a file called *ppflows* that contains a list of all process flow names that will be followed by lots in a release schedule. These process flows must already have been installed in the CAFE database. For more information on installing process flows in CAFE, please refer to [5]. Figure 4-2 shows a *ppflows* file containing two process flows, pfA and pfB. These flows are described by the Gantt charts in Figures 4-3 and 4-4. Flows pfA and pfB have 12 and 11 operations respectively, and both use the same three machine types M1, M2, and M3. Note that M2 and M3 only consist of one machine each, while M1 consists of two machines called M1-A and M1-B.

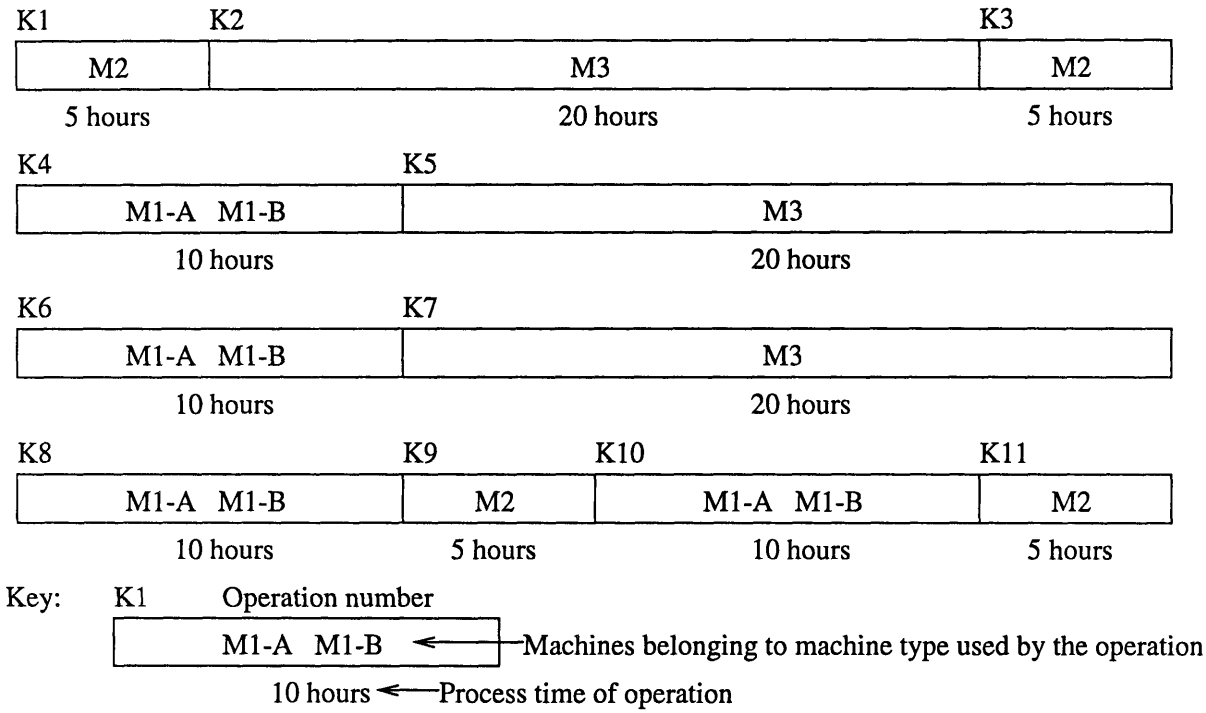


Figure 4-4: Gantt chart for process flow pfB

For each process flow listed, the module extracts from CAFE the operation time and the machine type used by each process step. The module then outputs this information into text files, one for each process flow. Figure 4-5 shows the text file created for flow pfA.

The Matrix generator module's purpose is to aggregate all necessary data and formulate the model described in Chapter 3 into a text file readable by GAMS. The primary input of the Matrix generator module is a file called *schedule* which contains

```
12
3
("m1-a" "m1-b")    6  1  36000  3  36000  5  36000  7  36000  9  36000  11  36000
"m2"               4  2  18000  4  18000  8  18000  10  36000
"m3"               2  6  72000  12  72000
```

Figure 4-5: Text file created by Preprocessor for flow pfA

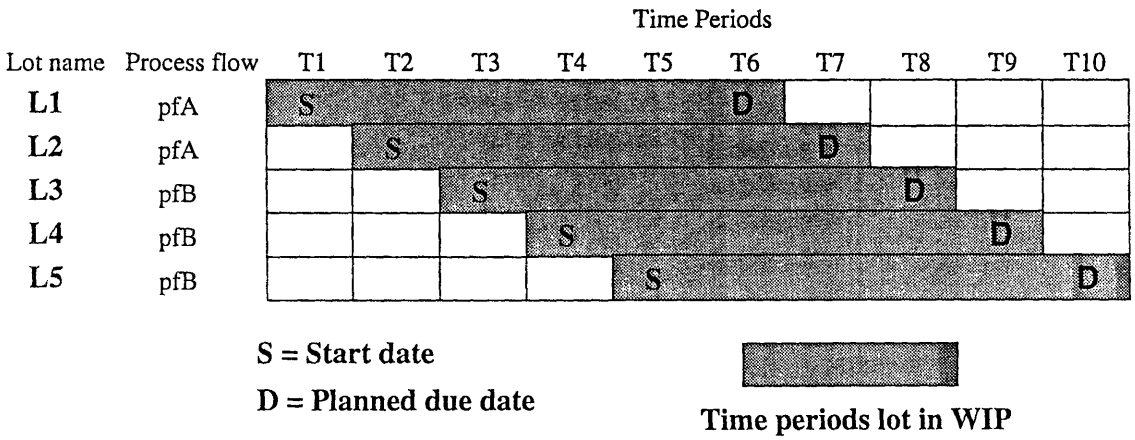


Figure 4-6: Sample Release Schedule

| LOT_NAME | PROC_FLOW | LEAD_TIME | START_DATE | DUE_DATE | SCHEDULE | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 |
| L1 | pfA | 6 | T1 | T6 | S | | | | | D | | | | |
| L2 | pfA | 6 | T2 | T7 | | S | | | | | D | | | |
| L3 | pfB | 6 | T3 | T8 | | | S | | | | | D | | |
| L4 | pfB | 6 | T4 | T9 | | | | S | | | | | D | |
| L5 | pfB | 6 | T5 | T10 | | | | | S | | | | | D |

Figure 4-7: Sample Release Schedule in Spreadsheet Format

the release schedule that the user wants to test. This file is created by any program that can export text spreadsheet files whose columns are delimited by tabs.

For example, to create the release schedule shown in Figure 4-6, a corresponding spreadsheet shown in Figure 4-7 was created. This spreadsheet needs to be in the format shown in Figure 4-7 so that it can be read by the Matrix generator. This format requires that the lot's process flow, and its start and planned due date are listed in the columns of their corresponding headers. Note that the start and due date information can be entered by either putting the time periods in the corresponding start and due date columns, or by placing *S*'s and *D*'s in the appropriate columns. Also note that start and due date information can also be inputted using either just the start date and lead time, or the due date and lead time. Lead time here is just the number of time periods between and including the start and due dates.

The Matrix generator module also takes as input a file called *crit_machine_list* as shown in Figure 4-8. This file contains a list of machine types the user believes to be

```
("m1-a" "m1-b")
```

Figure 4-8: Initial *crit_machine_list* file used for the toy example

```
MACHINE ANALYSIS TOOL MENU
**************************

    1) Preprocessor Module
    2) Matrix Generator Module
    3) Run Gams
    4) Analysis Module
    5) Check Critical Machine Assumptions
    6) Quit
Type number corresponding to menu option:2

 Input time (in hours) available each time period: 30

 Input utilization factor: 1

 Please just press enter when done changing machines
  Input name of machine type that you want to change number of
    and/or efficiency (instead of default):

 Critical machine ("m1-a" "m1-b") in LP is M1
```

Figure 4-9: Matrix generator user input

critical. Figure 4-8 shows that only machine type $M1$ is believed to be critical.

Finally, as shown in Figure 4-9 the Matrix generator asks the user to input the number of hours $N$ in a time period, and the desired utilization factor $\xi$. The Matrix generator also allows the user to change the number of machines in a machine type $E_m$, or its efficiency $e_m$. In this example, $N = 30$ and $\xi = 1$ is used. Also, for simplicity the default efficiency $e_m = 1$ is used for all machine types.

The Matrix generator module takes all the above inputs and formulates a mathematical problem based on the model discussed in Section 3.4.3. This mathematical problem is written in a text file called *gamsin* using the GAMS modeling language. Because of its length, the *gamsin* file generated for this example is shown in Appendix A.

GAMS is then called to solve the problem coded in *gamsin* using the simplex method. After GAMS has solved the problem, it returns which of the four possible outcomes described in Section 3.2.1 has occurred. As shown in Figure 4-10, an optimal

solution was found.

GAMS also returns a text file containing the problem's solution. If the problem is infeasible, GAMS returns the closest solution it could find to be feasible. This text file is in a format that is very difficult to read and understand. The purpose of the Analysis module is to translate this text file into a graphical format that can be more easily understood by the user. The Analysis module does this by extracting information from the GAMS output file and creating files that can be then read by XFIG. These files contain the following 3 pieces of information: 1) the probability that a critical operation will occur during a certain time period, 2) the expected time that a critical operation occurs, and 3) the expected load on each critical machine during each time period in the time frame.

Figures 4-11, 4-12, 4-13, and 4-14 show the XFIG files generated by the Analysis Module for the toy example as would be displayed by XFIG. Figure 4-11 shows the average loads on the machine types listed in the *crit_machine_list* file. Each shaded bar in the figure represents the average load during a time period on the machine type whose name is listed underneath. This average load is the same as that represented by the left side of Equation (3.15). Each unshaded bar represents the maximum capacity for a machine type and corresponds to the right hand side of Equation (3.15). Note that all units are in hours. Thus, for time period T1, the average load on machine type M1 is 30 hours while the maximum capacity is 60 hours[1].

Figure 4-12 shows the time a critical operation is expected to occur, or equivalently, the values of the $V_l^k$ variables returned by the model. For a lot, each vertical line represents when within the time frame the corresponding operation is expected to occur. Each vertical line is labelled by the operation number, the value of $V_l^k$, and the machine type the operation uses. For example, $V_l^k = 1$ for lot L1's operation K1 which uses machine type M1.

Figures 4-13 and 4-14 show the probability that critical operations occur during a particular time period. They display the primary decision variables $w_l^k(i)$ in the

[1]Since M1 consists of two machines, each one having 30 hours available per time period, the total number of hours on M1 per time period is 60.

```
MACHINE ANALYSIS TOOL MENU
**************************

    1) Preprocessor Module
    2) Matrix Generator Module
    3) Run Gams
    4) Analysis Module
    5) Check Critical Machine Assumptions
    6) Quit
Type number corresponding to menu option:3
going to make directory GG15504
--- gams 2.05
--- compiling then executing /amd/hierarchy/c/singzhen/lp/gamsin
--- output is being written to /amd/hierarchy/c/singzhen/lp/gamsin.lst
--- starting Minos 5.2


 ==============================
 M I N O S    5.2    (Mar 1988)
 ==============================

 BEGIN GAMS/MINOS options
READING DATA...
 WORK SPACE NEEDED (ESTIMATE)  --   13945 WORDS.
 WORK SPACE AVAILABLE          --   16735 WORDS.

   Itn  Nopt  Ninf  Sinf,Objective
     1     1    53  1.61969425E+02
   100     3    20  1.20059002E+01
   200     7     2  8.54130990E-02

 Itn   210 -- Feasible solution.  Objective =   3.269035088E+01

 Itn   232 --     22  nonbasics set on bound, basics recomputed

 Itn   232 -- Feasible solution.  Objective =   3.150000000E+01


 EXIT -- OPTIMAL SOLUTION FOUND

 Major, Minor itns            1         232
 Objective function      3.1500000000000E+01
 Degenerate steps            96       41.38
 Norm X,      Norm PI    4.56E+00    1.78E+01
 Norm X,      Norm PI    3.99E+01    2.19E+00  (unscaled)
--- resuming execution
--- gams (15504) done
```

Figure 4-10: Output displayed by GAMS

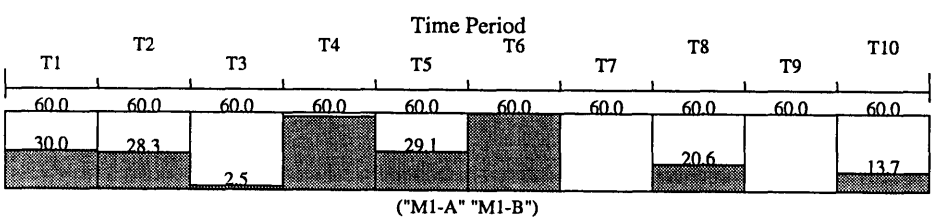


Figure 4-11: XFIG display of average loads on critical machine types returned for toy example

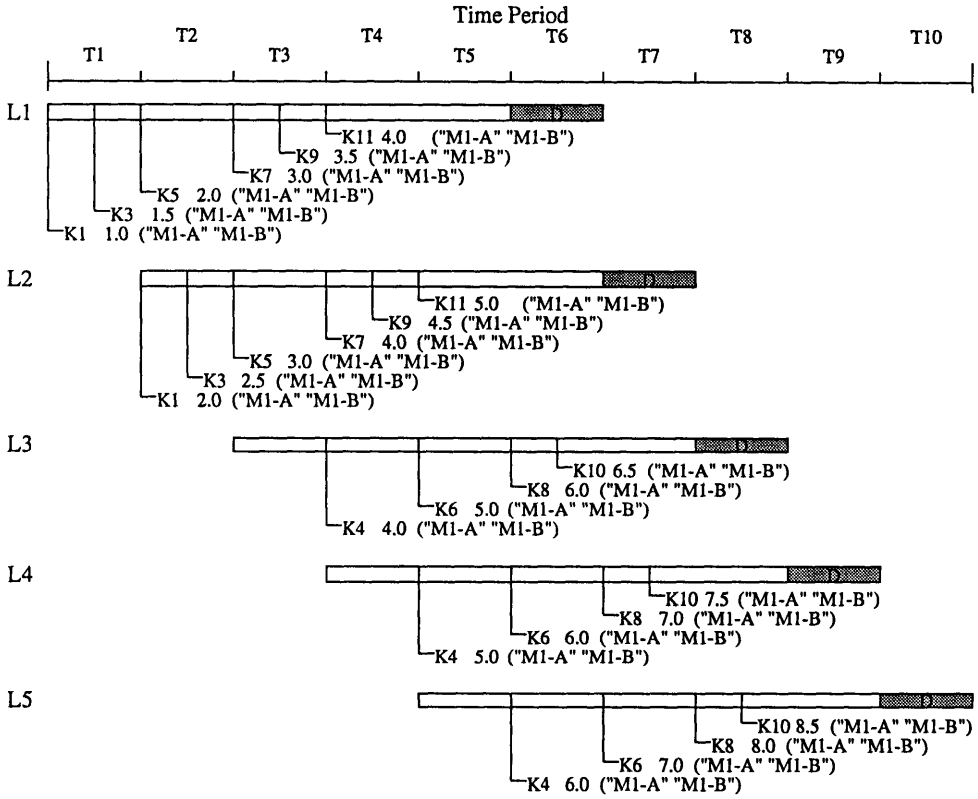


Figure 4-12: XFIG display of $V_l^k$ returned for toy example

model. For each critical operation of each lot, the value of $w_l^k(i)$ is displayed on top of a shaded bar. How high each bar is depends on how large the value of $w_l^k(i)$ is. On the left hand side also displays the operation number as well as the machine type used by the lot. Thus, for lot L1's operation K1, $w_l^k(i) = 1$ for time period T1 and zero for all other time periods.

As can be seen from Figure 4-11, the average load on machine type M1 is not particularly close to maximum capacity. However, it is necessary to check to see if the assumption that non-critical machine types have ample capacity is valid. The purpose of the Assumption Checker Module is to check these assumptions. If the assumptions are wrong, the list of critical machines is updated and the tool must be rerun. Figure 4-15 shows the results of running the Assumption Checker Module on this toy example. As can be seen, machine type M3's estimated load has exceeded its maximum capacity, thus violating its assumption as a non-critical machine. Thus, the Assumption Checker updates the *crit_machine_list* file by including M3 in the list.

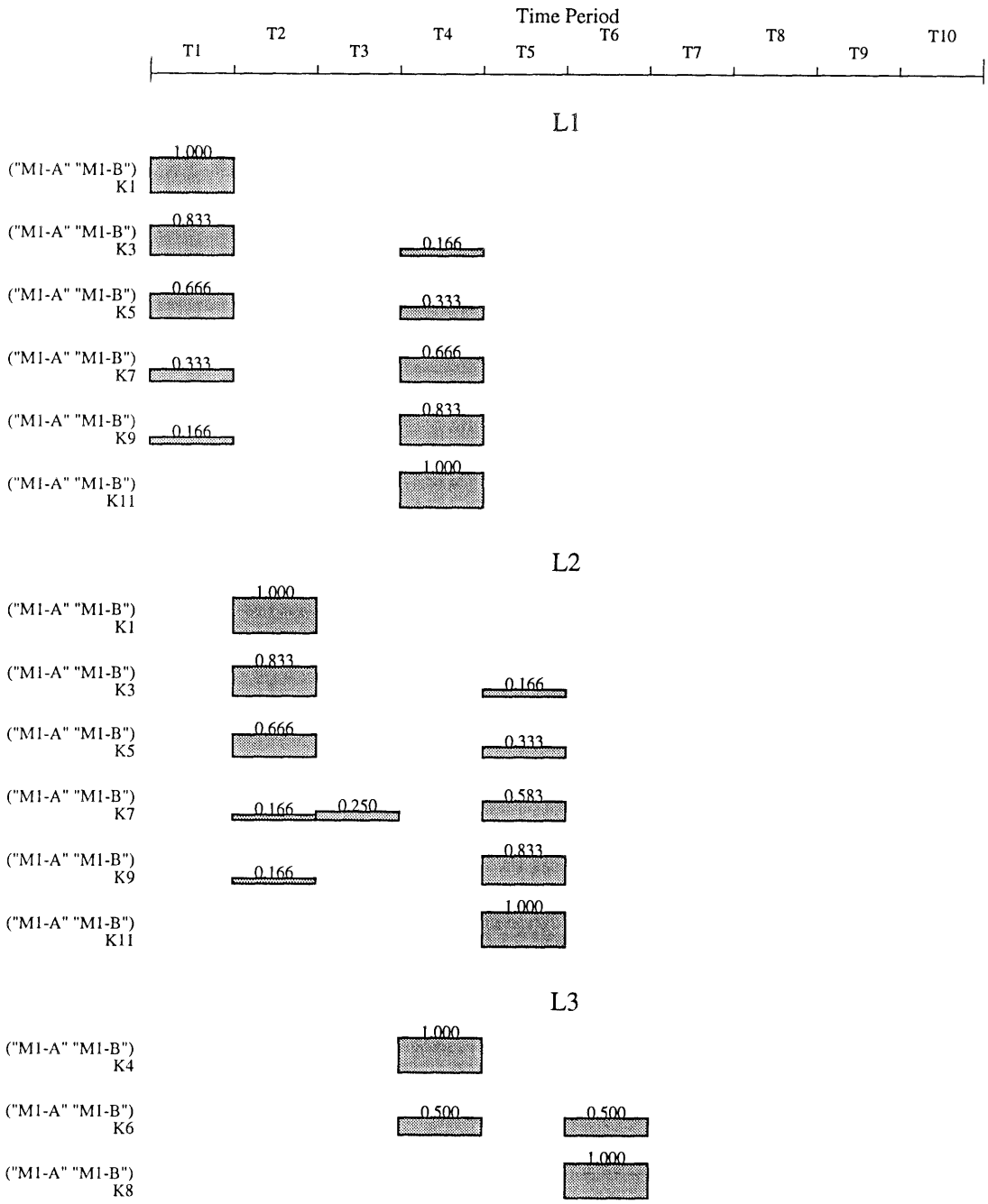


Figure 4-13: XFIG display (1) of $w_l^k(i)$ returned for toy example

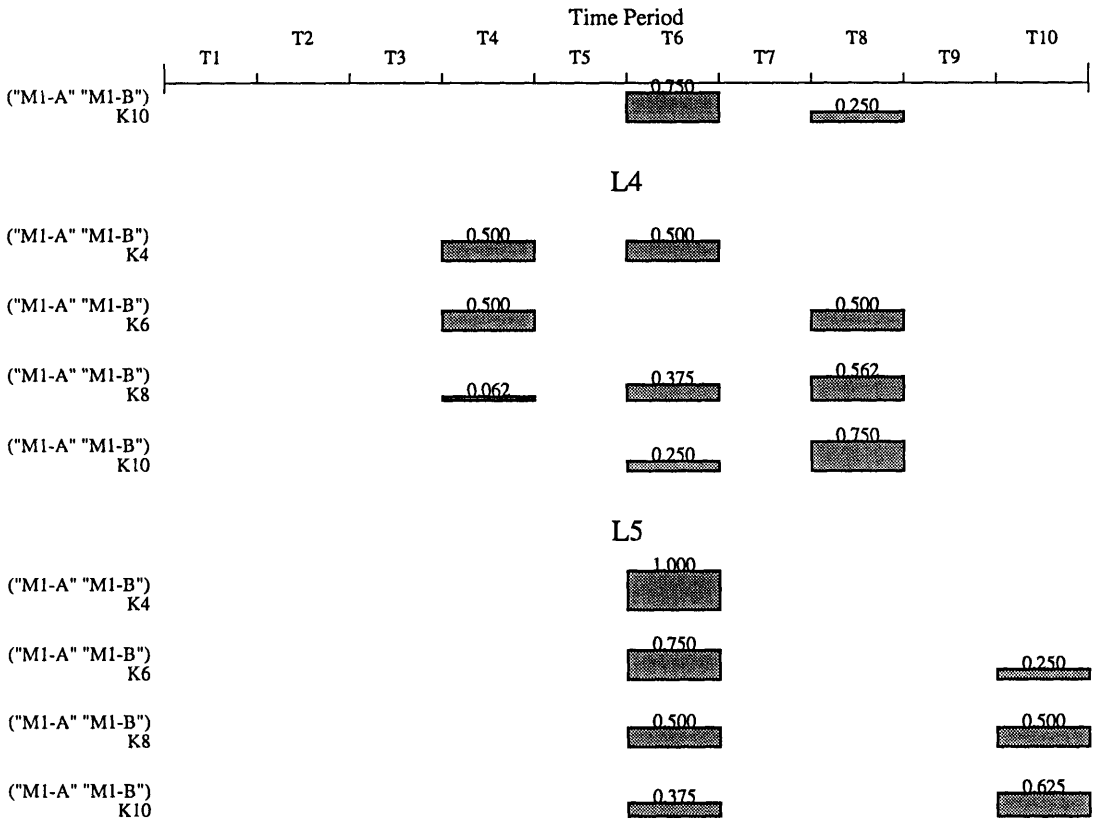


Figure 4-14: XFIG display (2) of $w_l^k(i)$ returned for toy example

```
MACHINE ANALYSIS TOOL MENU
**************************

    1) Preprocessor Module
    2) Matrix Generator Module
    3) Run Gams
    4) Analysis Module
    5) Check Critical Machine Assumptions
    6) Quit
Type number corresponding to menu option:5
MACHINE "m3" has violated non-critical machine assumption on time period 2
     cap_up=30.000000  <  cap_l=40.000000
MACHINE "m3" has violated non-critical machine assumption on time period 3
     cap_up=30.000000  <  cap_l=40.000000
MACHINE "m3" has violated non-critical machine assumption on time period 4
     cap_up=30.000000  <  cap_l=60.000000
MACHINE "m3" has violated non-critical machine assumption on time period 5
     cap_up=30.000000  <  cap_l=60.000000
MACHINE "m3" has violated non-critical machine assumption on time period 6
     cap_up=30.000000  <  cap_l=40.000000

 Do you wish to create xfig files of cap_l for all machines?
     Type n for no, anything else for yes :
CURRENTLY PRINTING FILE gamsin.lst_all_cap_l1.fig

 Do you wish to update crit_machine_list?
     Type n for no, anything else for yes :

 Do you wish to include all machines that violated critical machine assumption?
                    Type n for no, anything else for yes :

 Do you wish to keep previous critical machine ("m1-a" "m1-b") having
     Type n for no, anything else for yes :
```

Figure 4-15: Results of the Assumption Checker Module on the toy example

The tool is then rerun.

The results of rerunning the tool with the same $N$, $\xi$, and $e_m$'s are shown in Figures 4-16, 4-17, 4-18, 4-19, and 4-20. Running the Assumption Checker Module on these new results resulted in no violations in any assumptions on non-critical machines. However, note that the load on machine type M3 is close to its maximum capacity. As explained in Chapter 3, when loads on machines are close to maximum capacities, the problem's feasibility becomes highly vulnerable to disruptions.

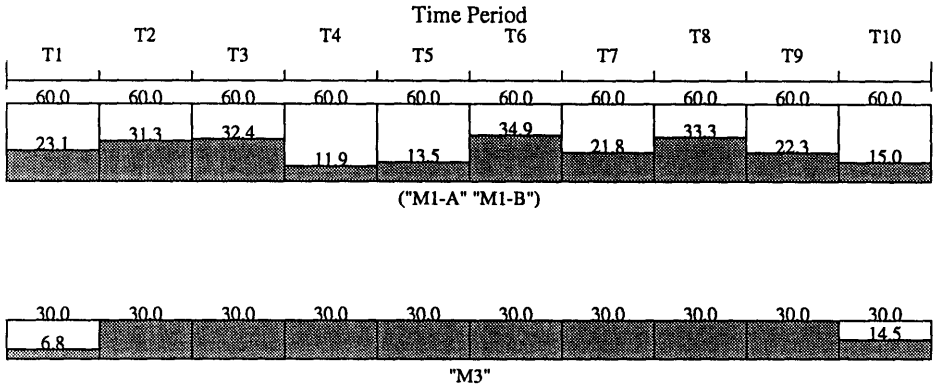


Figure 4-16: Average loads on critical machine types after including M3 as a critical machine type

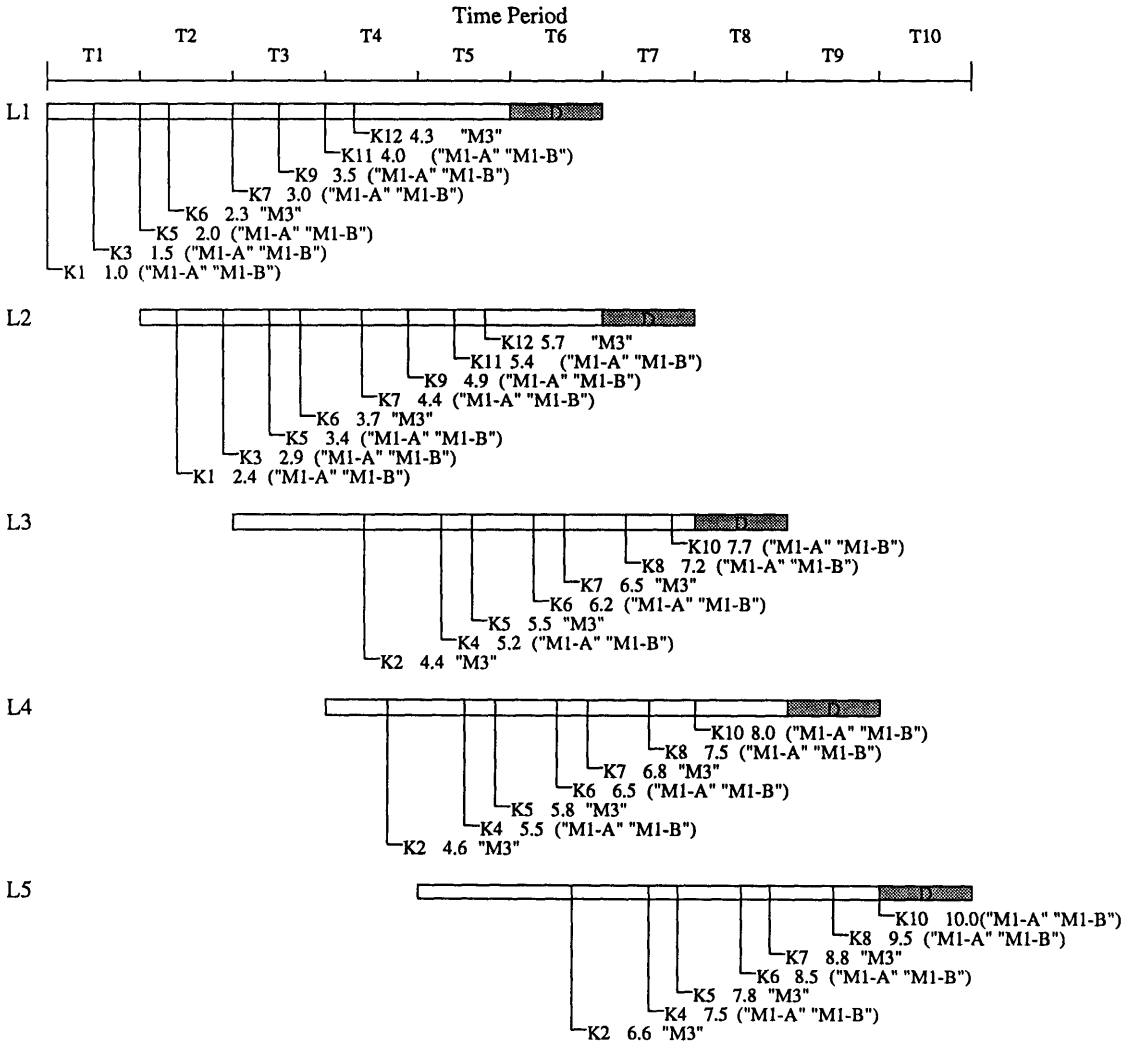


Figure 4-17: $V_l^k$ after including M3 as a critical machine type

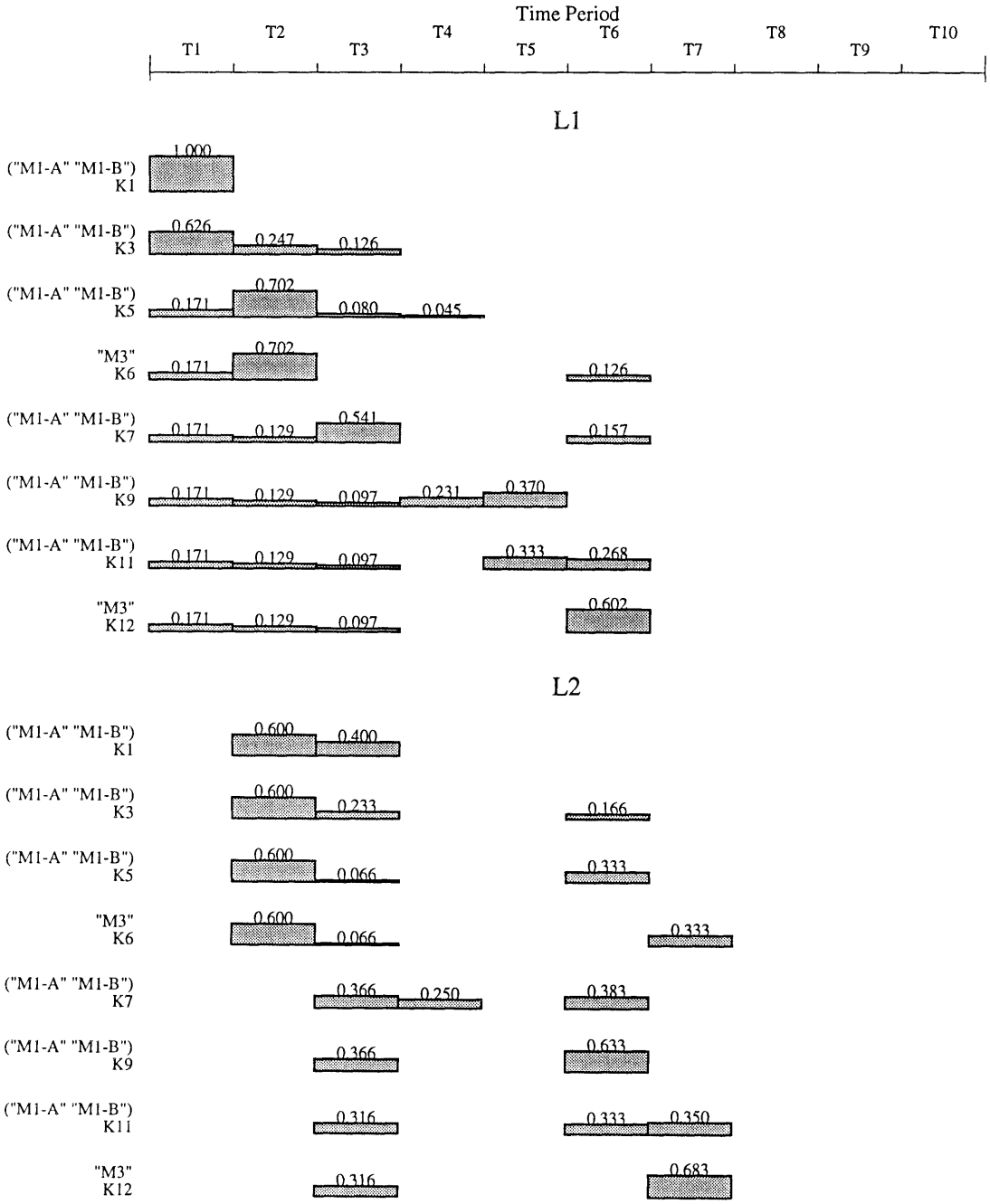


Figure 4-18: $w_l^k(i)$ after including M3 as a critical machine type

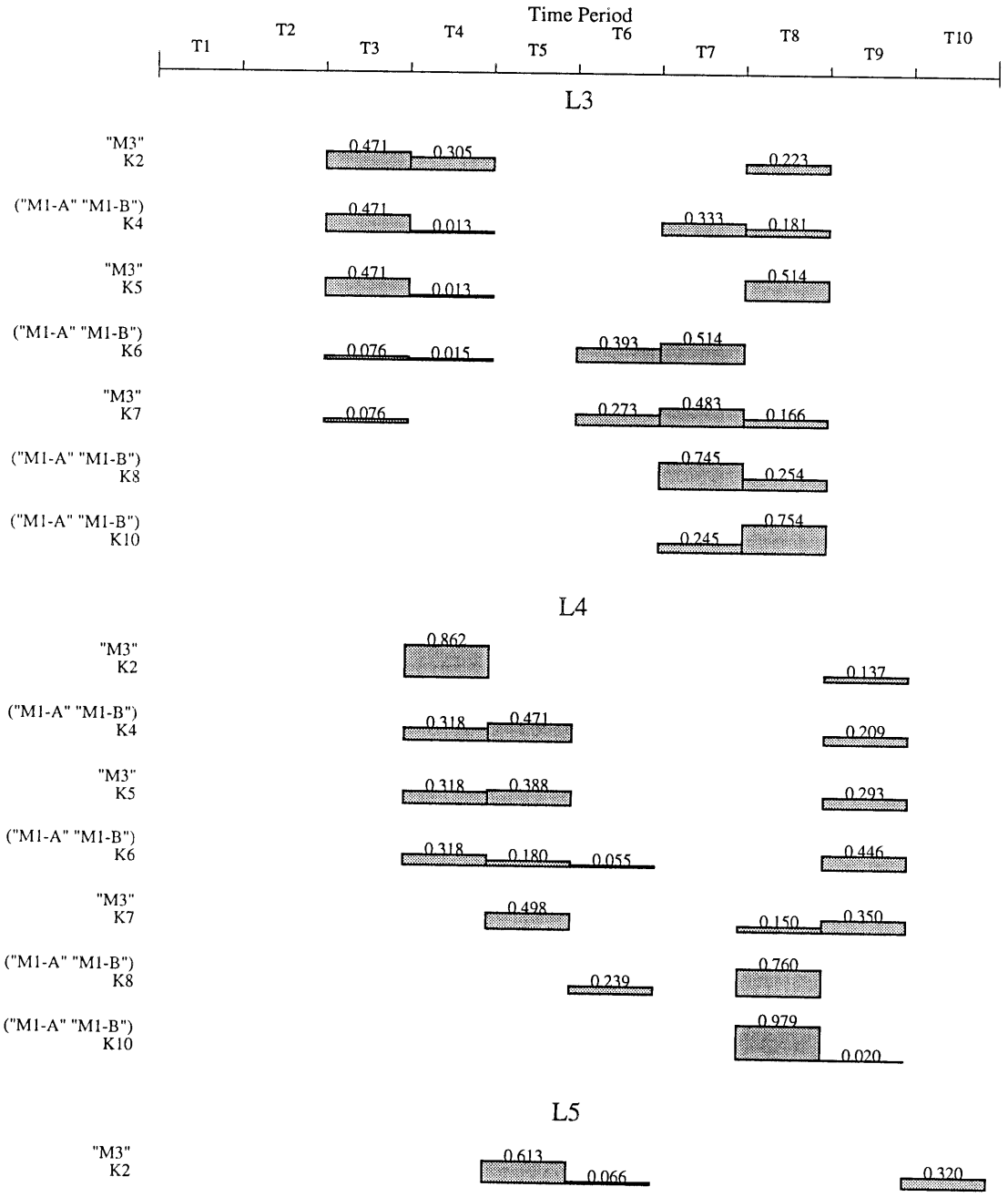


Figure 4-19: $w_l^k(i)$ after including M3 as a critical machine type

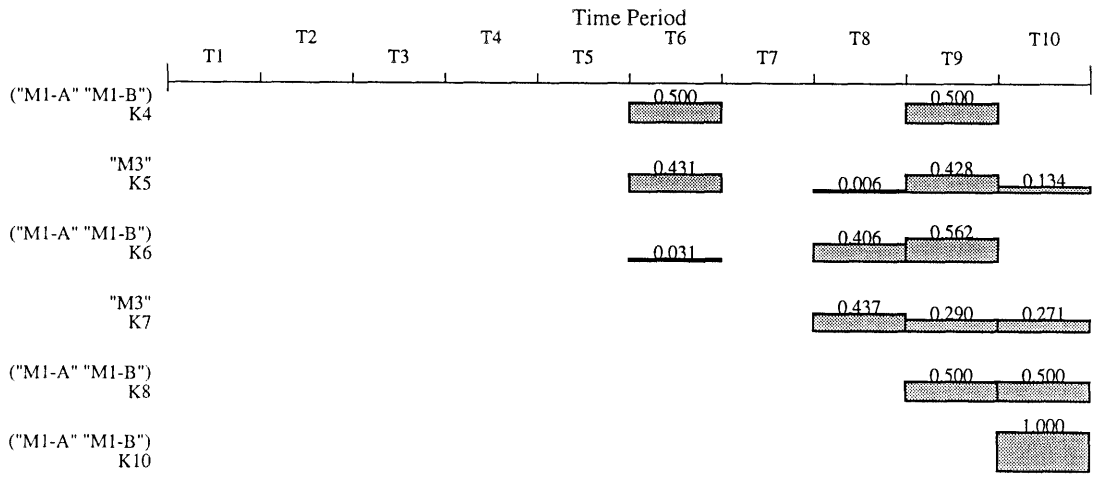


Figure 4-20: $w_l^k(i)$ after including M3 as a critical machine type

# Chapter 5

# Model Behavior and Complexity

This chapter first describes the model's qualitative behavior as capacity increases, and shows that this behavior agrees with our intuition about how reality behaves. Next, the validity of the critical machine assumption as a function of capacity is explored. Finally, this chapter discusses the computational burden required to solve the model as problem size increases.

## 5.1 Behavior as Capacity Increases

When a manufacturing system is operating near full capacity, it is very difficult to do everything in an ideal manner. If machine loads are so great that only a few loading schedules can deliver products by their due dates, then there is little freedom to choose desirable schedules. Therefore, we would suspect that if capacity increases, a better loading schedule can be obtained. We now show that the model described above also exhibits this behavior.

Figures 4-16, 4-17, 4-18, 4-19, and 4-20 showed the results of a system operating very close to its maximum capacity. For these figures, an objective function was chosen such that the expected time of the last critical operation $V_l^{last}$ of each lot would be minimized. As a result, the linear program tried to to find the set of $w_l^k(i)$ such that the expected times operations occurred were as close to the start dates as possible.

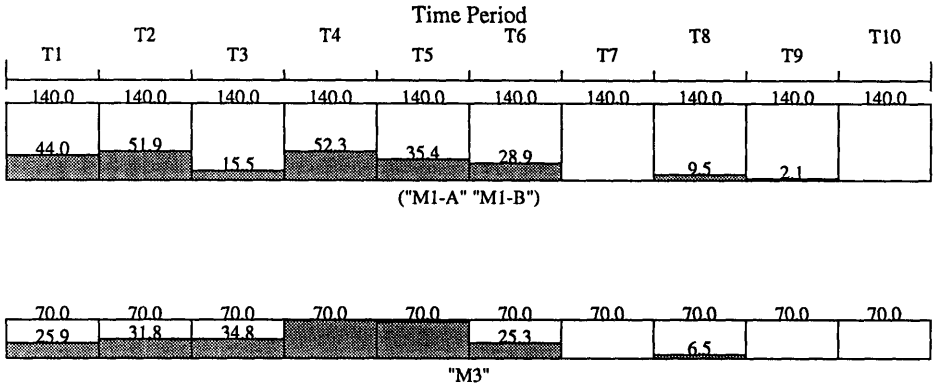


Figure 5-1: Average loads on critical machine types, $N = 70$

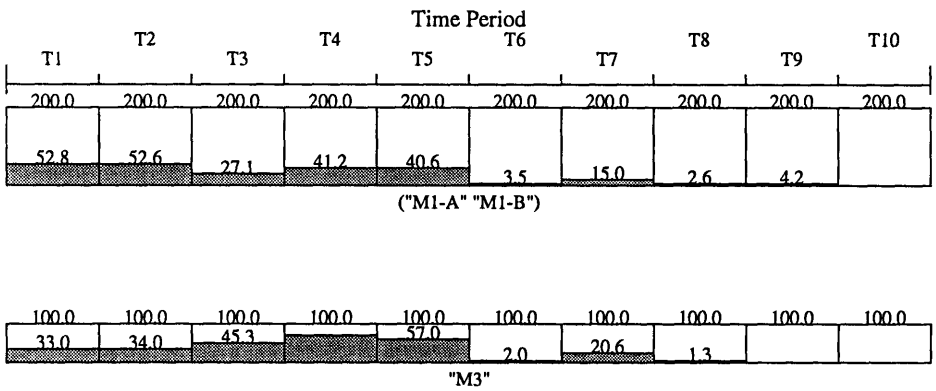


Figure 5-2: Average loads on critical machine types, $N = 100$

Figures 5-1 and 5-4 show the results of increasing the number of hours in a time period $N$ on the same problem to 70. Doing so has the effect of increasing the maximum capacity of the system. Figures 5-2 and 5-5 and Figures 5-3 and 5-6 show what happens when $N$ is further increased to 100 and 200 respectively. As can be seen, as the maximum capacity increases, the expected times operations occur are closer to the start dates, thus agreeing with our intuition.

It is important to remember that the model's objective function need not correspond to how fab management reacts to various changes. Fab management's first reaction to an increase in capacity may be to take things a little easier and not load operations as early as possible. Since there is no closed feedback loop from reality back to the model, this reaction would not be taken into account.

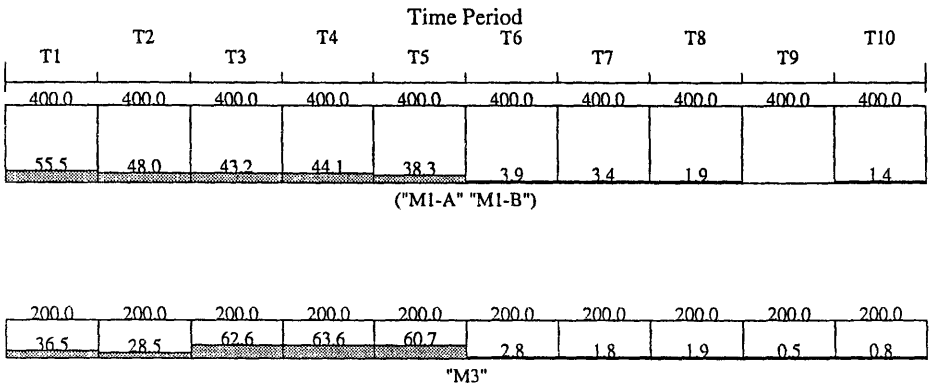


Figure 5-3: Average loads on critical machine types, $N = 200$

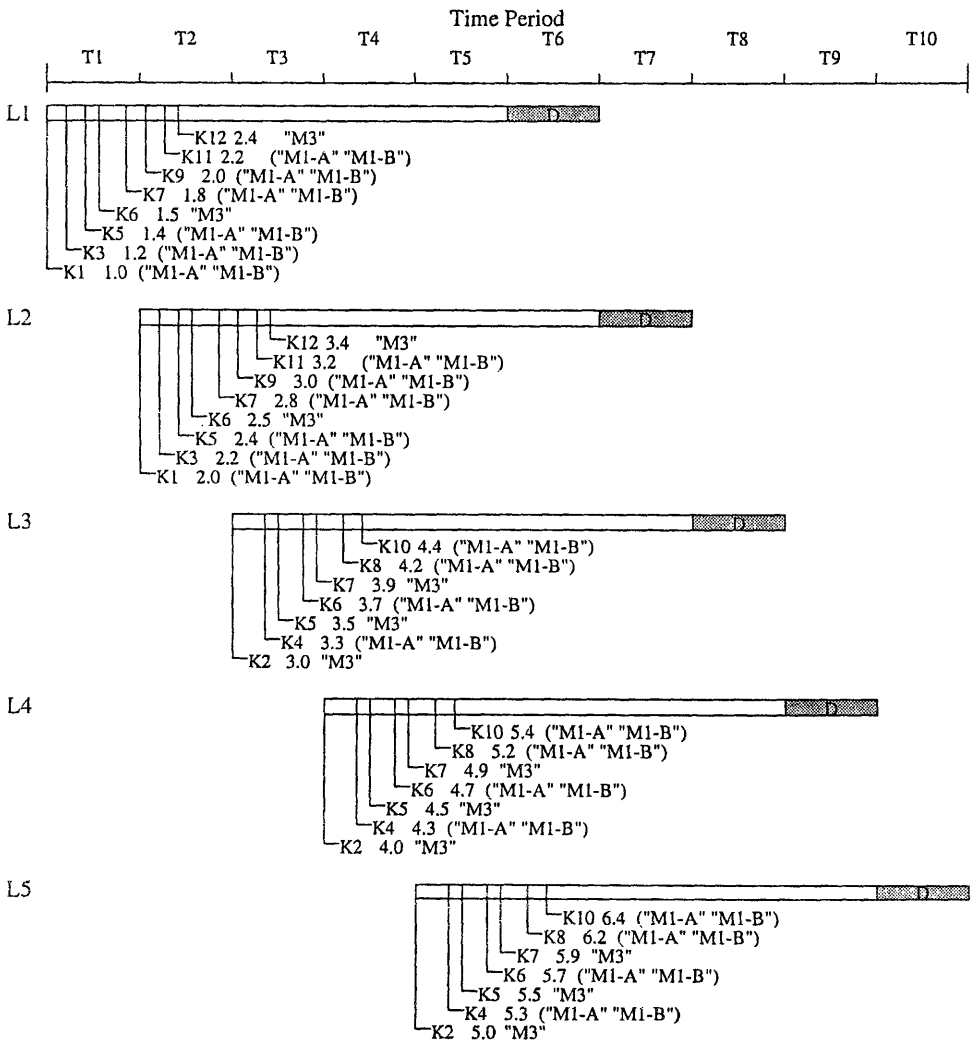


Figure 5-4: $V_l^k$ for critical operations, $N = 70$

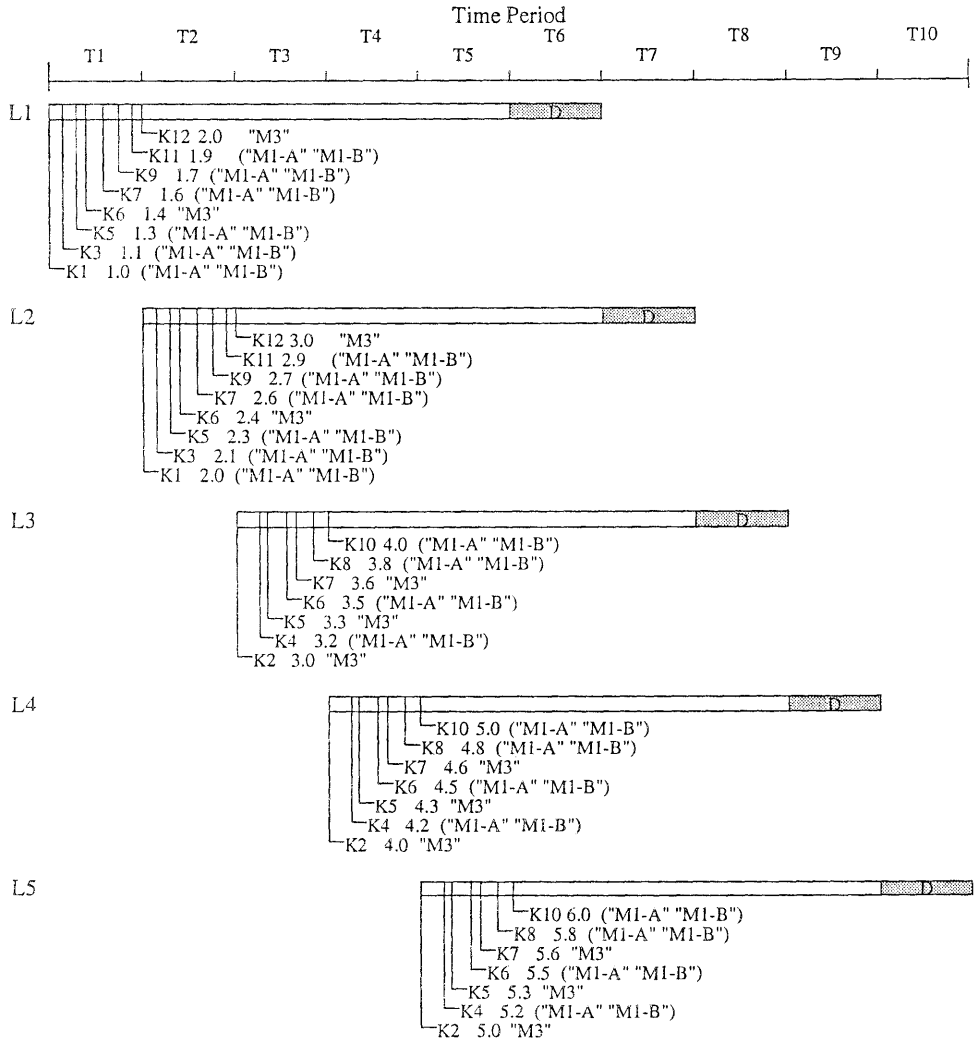


Figure 5-5: $V_l^k$ for critical operations, $N = 100$

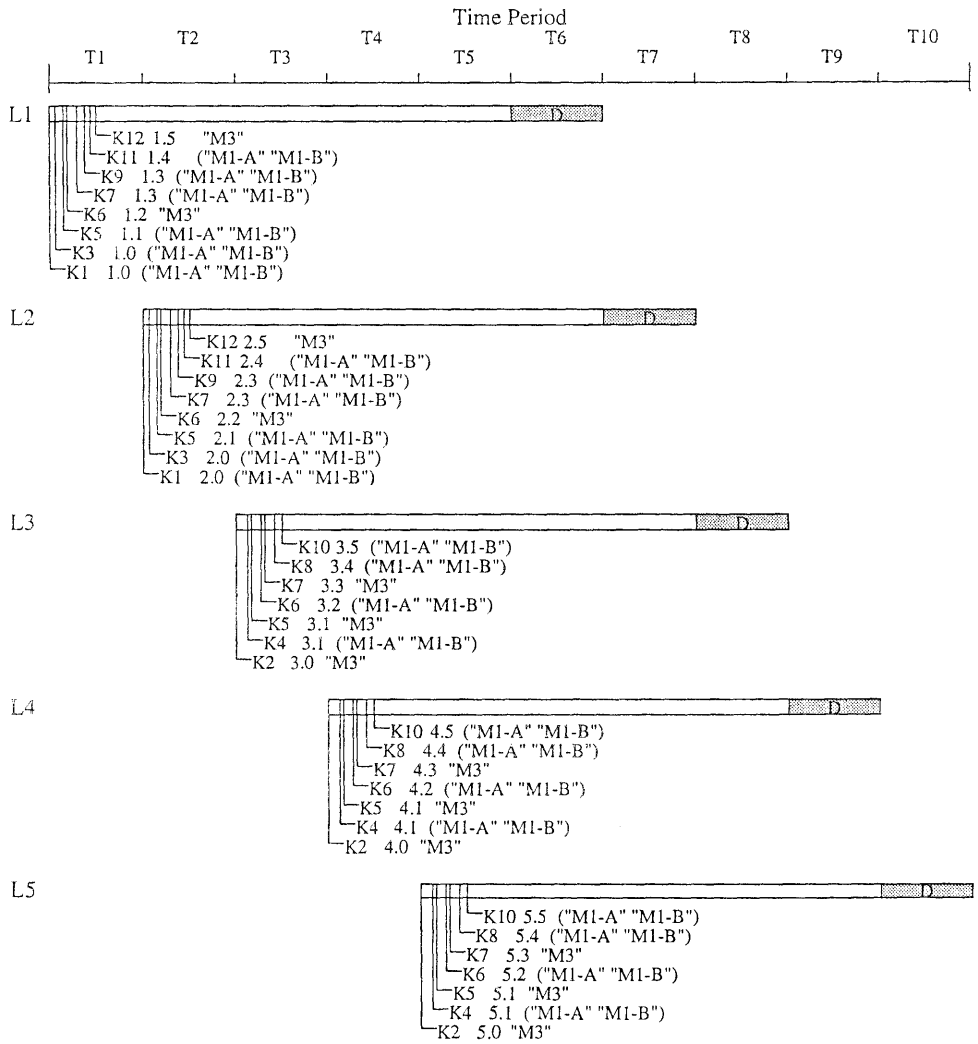


Figure 5-6: $V_l^k$ for critical operations, $N = 200$

## 5.2 Critical Machine Assumption Validity

The selected focus technique used to simplify the original binary ILP problem relied upon the assumption that the solution would not be affected greatly by ignoring non-critical machines from the formulation. However, as discussed before, the model's various assumptions and simplifications break down as the manufacturing system gets closer to full capacity. Here we show the degree of error resulting from ignoring non-critical machines, as a function of how close the system is to full capacity.

Figures 5-7, 5-11, and 5-12 are the results of the same problem solved in Section 4.2. This time however, the non-critical machine type M2 is included as a critical machine. As a result, an infeasible solution results, as can be seen in Figure 5-12 where the expected time of lot 5's second operation is after that of the third operation. Why did this problem's infeasibility lie undetected until now? The reason is that M2 was ignored from the formulation when it should not have been. M2 must be included in the formulation if the system operating so close to maximum capacity and the critical machine assumption is violated. This also confirms the suspicion aroused earlier that this problem was actually infeasible.

Figures 5-8, 5-13, and 5-14 show what happens when $N$ is increased to 70. Although a feasible solution is found this time, including M2 as a critical machine still adds some error to the solution. This can be observed by comparing Figures 5-8, 5-13, and 5-14 with Figures 5-1 and 5-4 which only had M1 and M2 as critical machine types.

This error diminishes greatly when the system is operating further and further away from maximum capacity. Figures 5-9, 5-15, and 5-16 and Figures 5-10, 5-17, and 5-18 show the results of $N$ being increased to 100 and 200 respectively. The results in these figures are close to those of Figures 5-2 and 5-5 and Figures 5-3 and 5-6 which had ignored M3 from the formulation. Therefore, we can see that the farther away a manufacturing system is from full capacity, the less error results from ignoring non-critical machines.

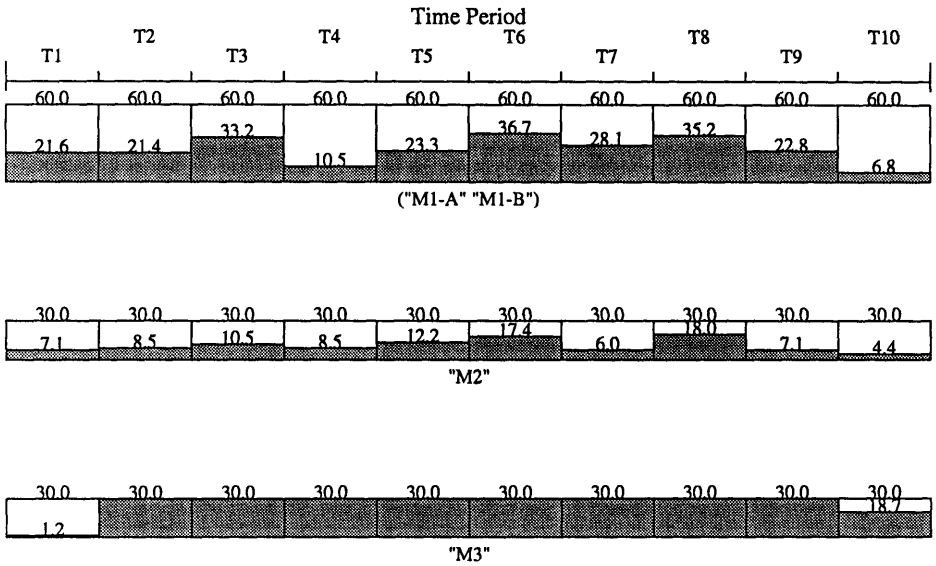


Figure 5-7: Average loads on all machine types, $N = 30$

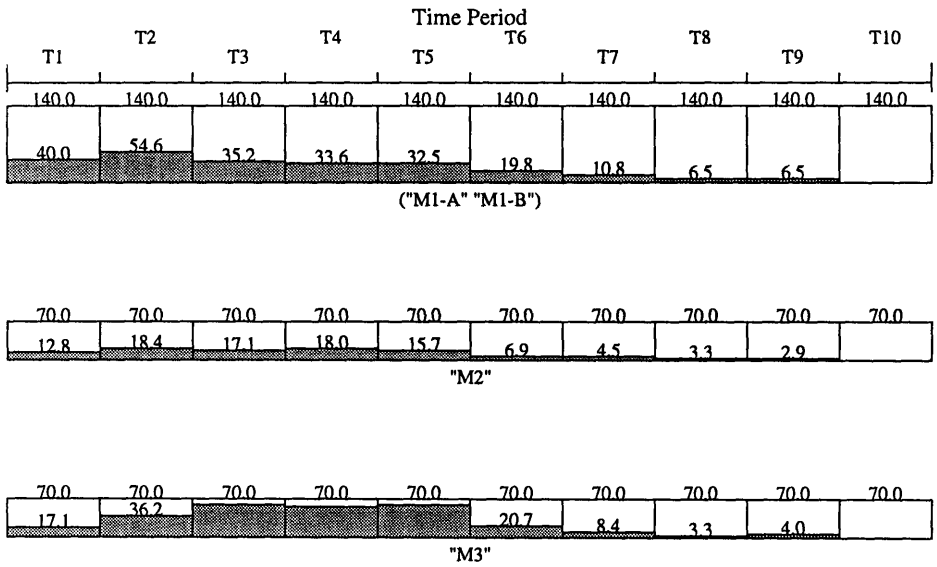


Figure 5-8: Average loads on all machine types, $N = 70$

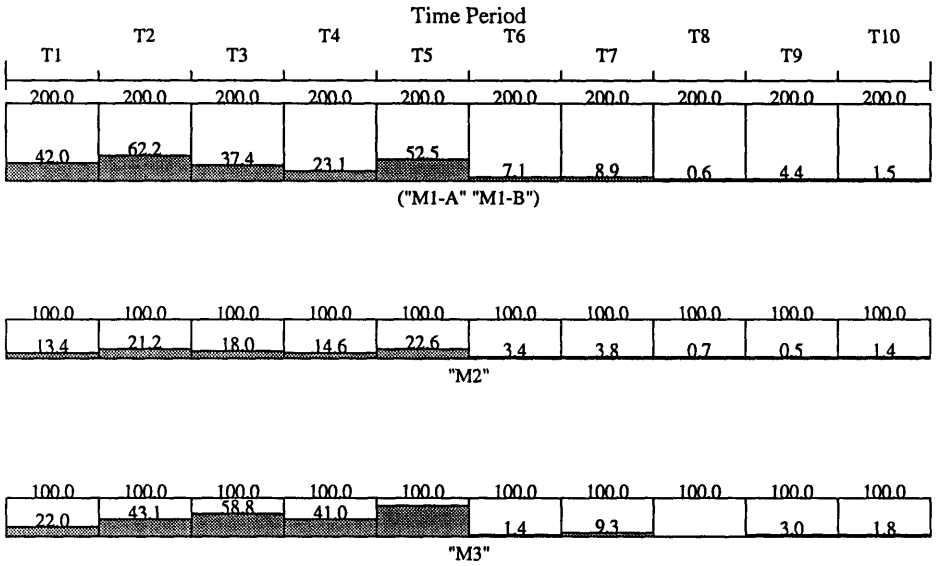


Figure 5-9: Average loads on all machine types, $N = 100$

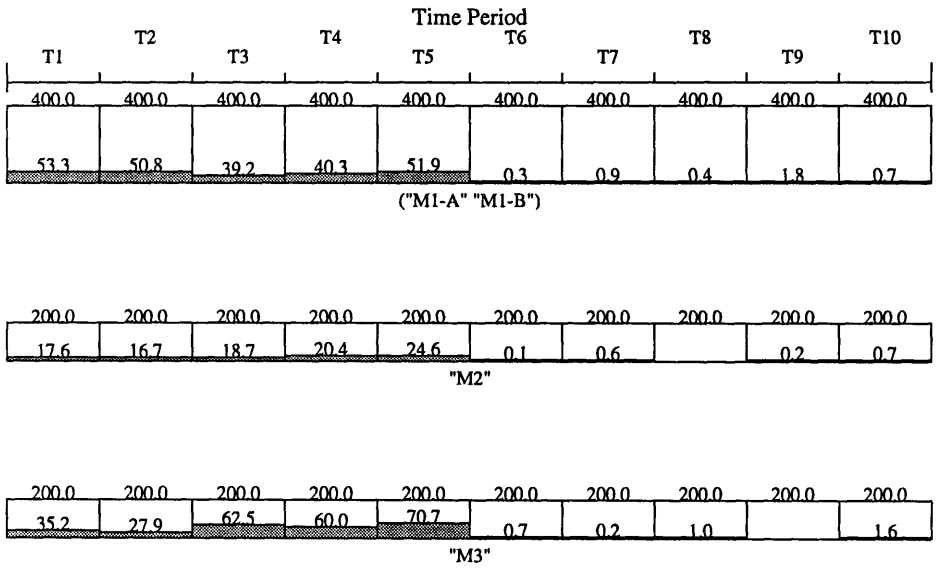


Figure 5-10: Average loads on all machine types, $N = 200$

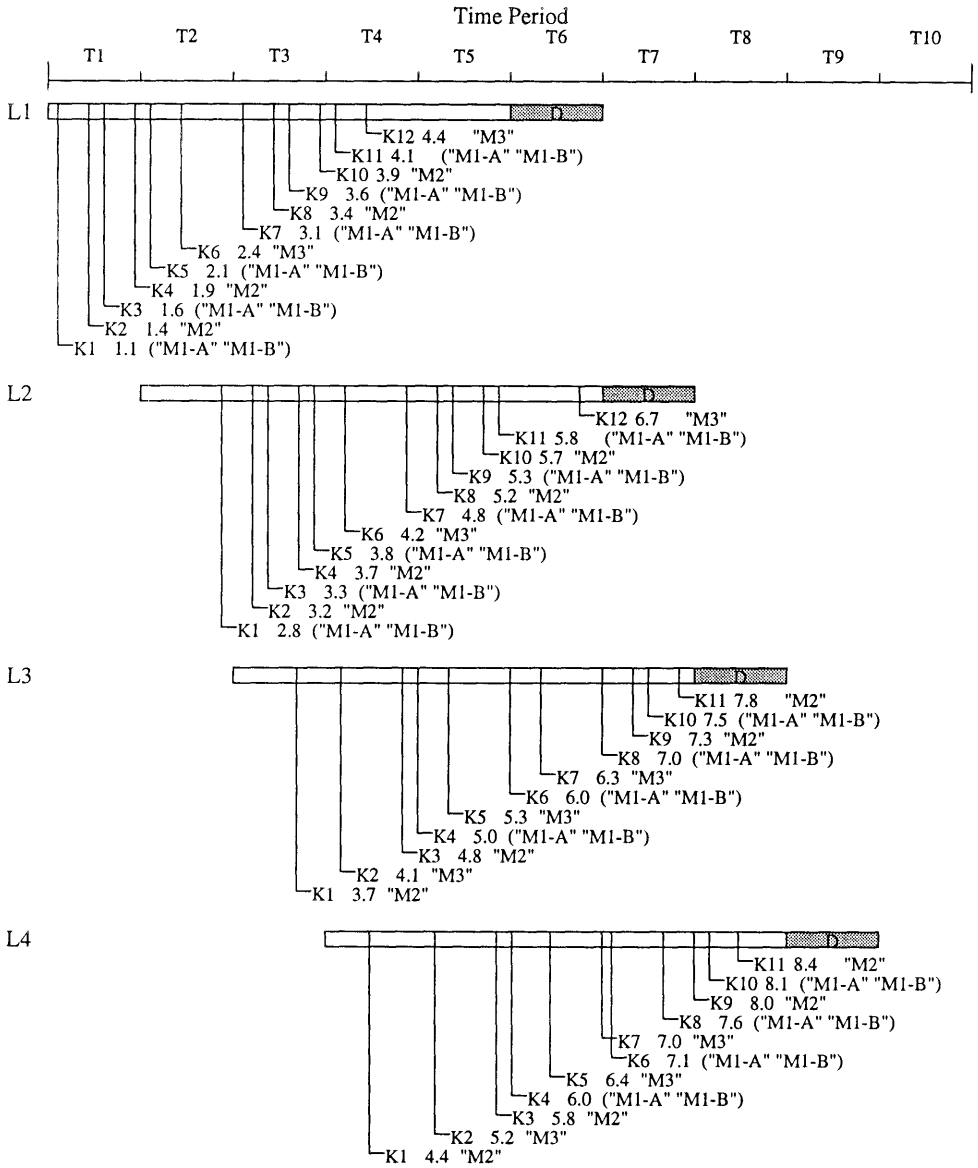


Figure 5-11: $V_l^k$ for all machine types, $N = 30$ (1)

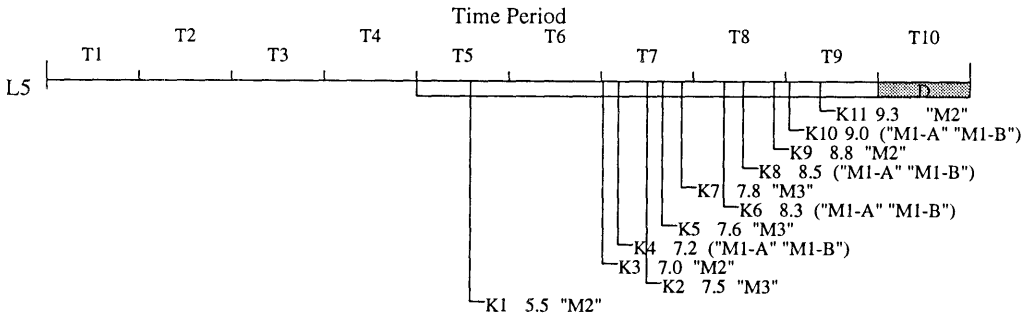


Figure 5-12: $V_l^k$ for all machine types, $N = 30$ (2)

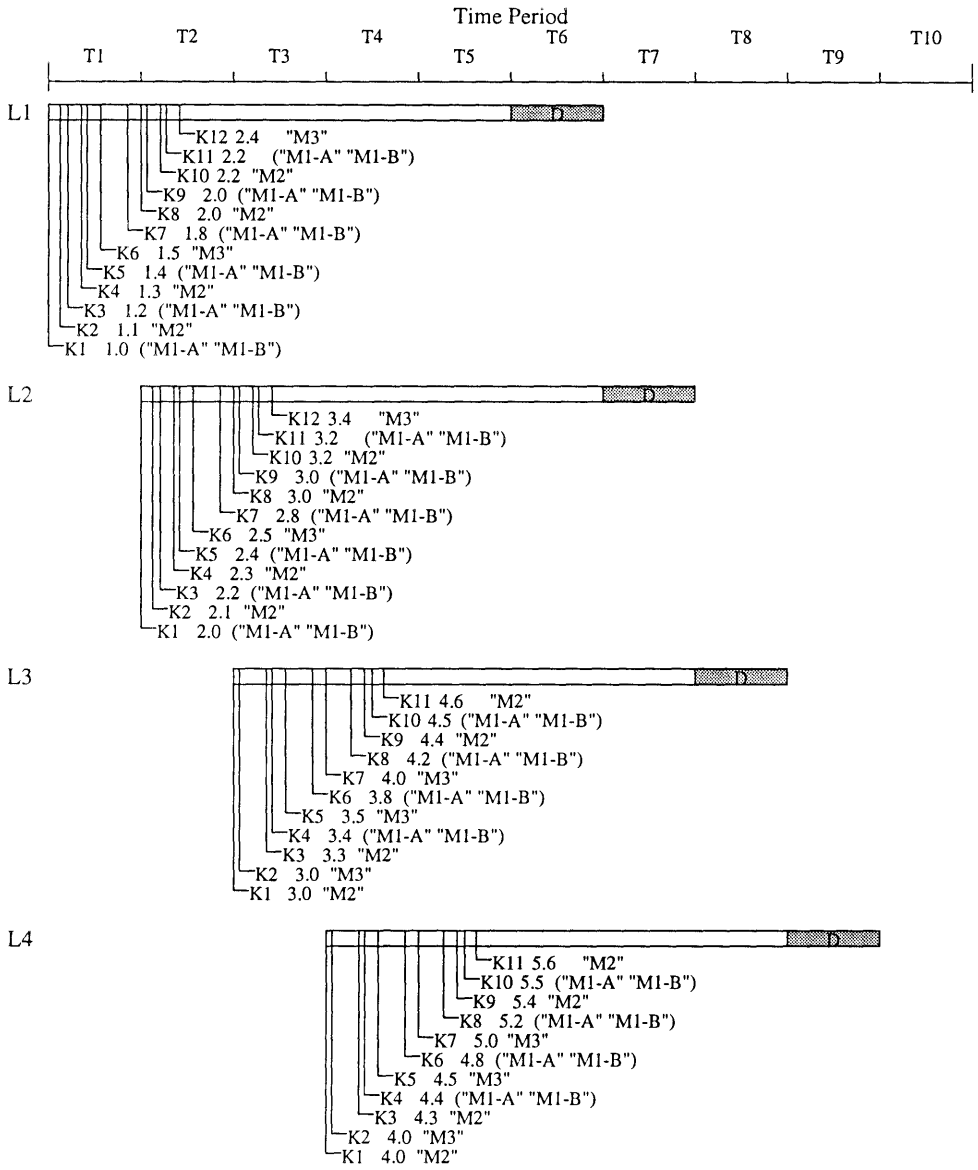


Figure 5-13: $V_l^k$ for all machine types, $N = 70$ (1)

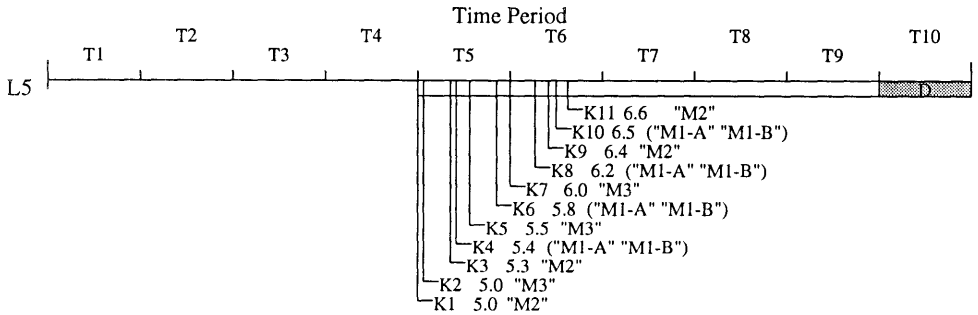


Figure 5-14: $V_l^k$ for all machine types, $N = 70$ (2)

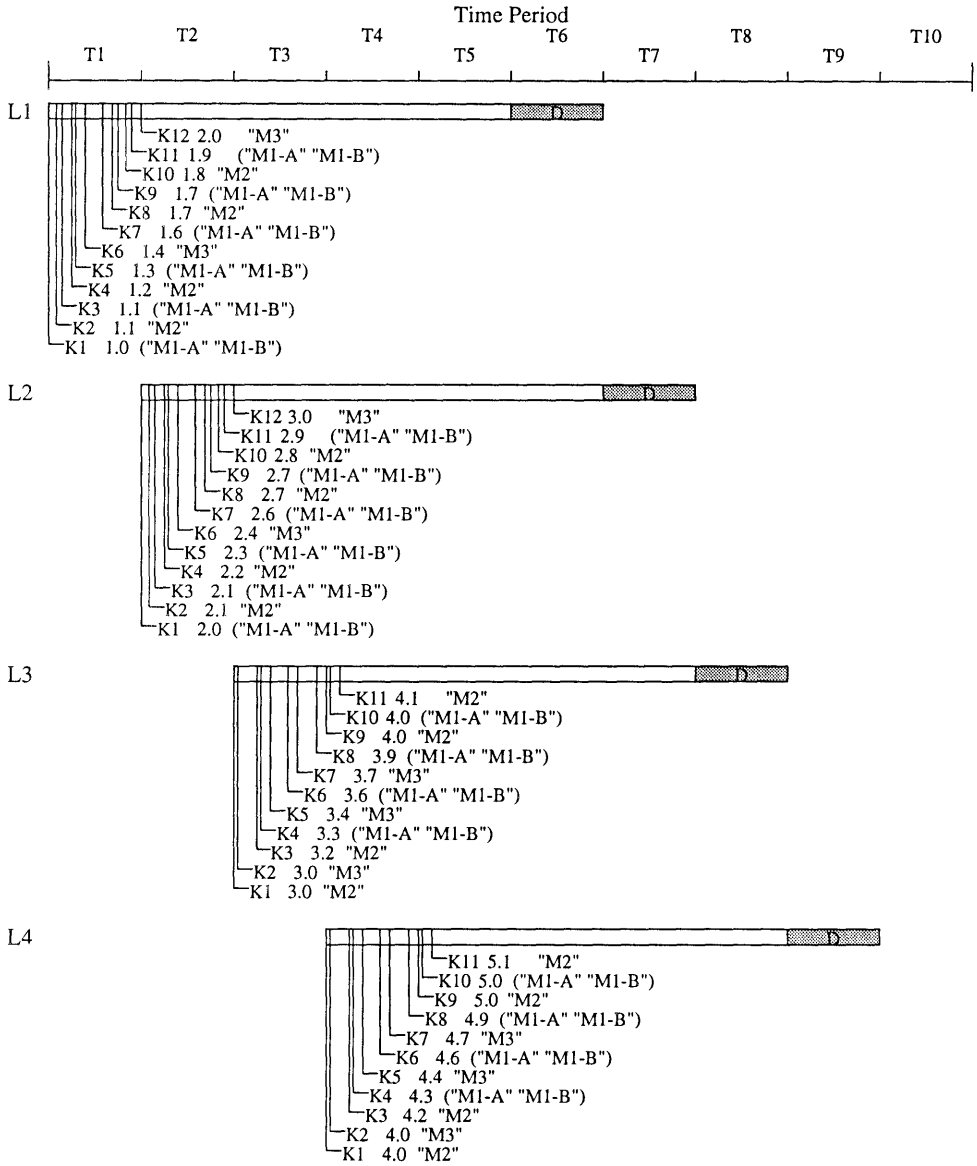


Figure 5-15: $V_l^k$ for all machine types, $N = 100$ (1)

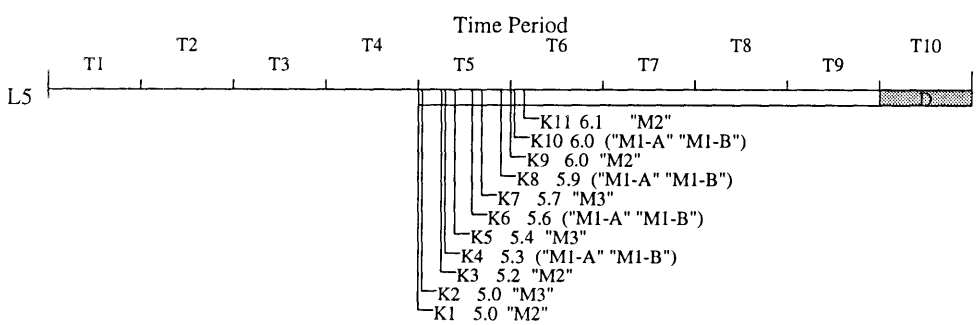


Figure 5-16: $V_l^k$ for all machine types, $N = 100$ (2)

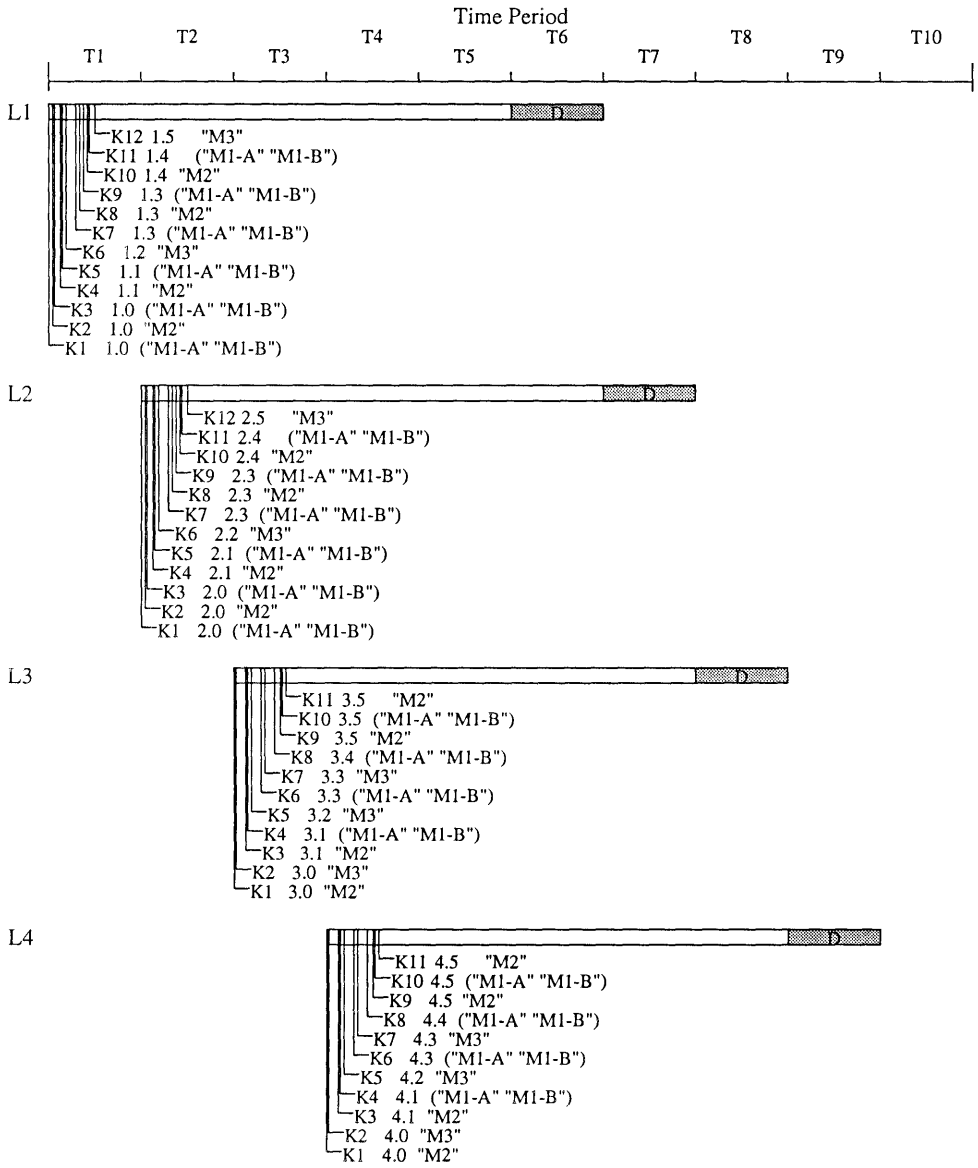


Figure 5-17: $V_l^k$ for all machine types, $N = 200$ (1)

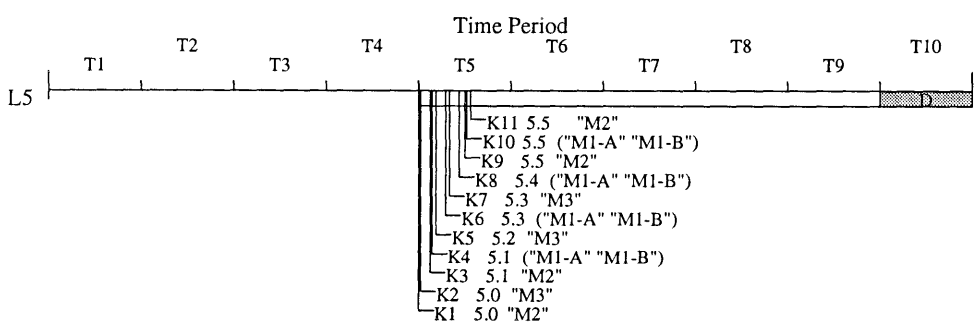


Figure 5-18: $V_l^k$ for all machine types, $N = 200$ (2)

## 5.3 Computational Issues

The techniques used in Section 3.4 were able to transform a previously intractable problem into a much smaller, easily solvable problem. However, for large problems, the memory requirements and computational burden of solving the problem can still be quite large. The following show the results of increasing the problem size by increasing number of lots, time periods, and critical operations. It should be remembered that the simplex method was used to solve the LP problem. Although the simplex method usually works well for small to medium sized problems, it can be very slow for very large problems. Newer methods which use interior point algorithms are much better suited to solve such large sized problems [4].

Figure 5-19 shows the size of a problem as a function of number of lots $L$ and the maximum number of critical operations $K$. The size of the problem is expressed in terms of how many *non-zero elements* there are. The number of non-zero elements are the number of non-zero coefficients in the matrix $A$ if Equation (3.24) was written in the matrix notation of Equation (3.2). Note in Figure 5-19 that the number of time periods and critical machine types are kept constant at 10 and 2 respectively. Note that increasing the number of lots or critical operations increases the problem size linearly.

However, the additional CPU time needed to solve the problem as the problem size gets bigger is extremely non-linear. Figure 5-20 shows the resource utilization in CPU seconds as a function of $L$ and $K$ again. Note that the vertical axis is on a logarithmic scale. As can be seen, the computation time needed to solve the problem increases rapidly as problem size increases. Because this could lead to difficulties with very large models, it is advised to keep problem size down to a minimal using the problem reduction techniques discussed before.

Figures 5-21 and 5-22 show similar results where this time the number of time periods $I$ is varied while the number of critical operations and machines are kept at 8 and 2 respectively. It should be noted that the machine loads for the problems sampled in Figure 5-20 and 5-22 were kept at a certain level relative to the system's

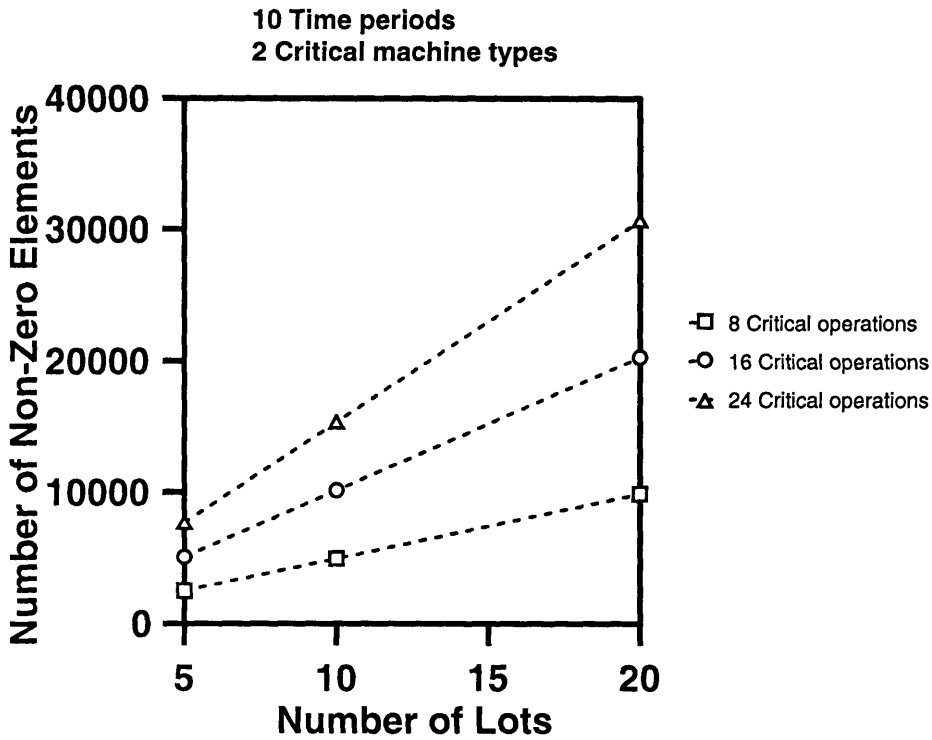


Figure 5-19: Problem Size as a Function of $L$ and $K$

maximum capacity. This way, there is less skew in the data arising from one problem requiring more CPU time because it is relatively more difficult to solve.

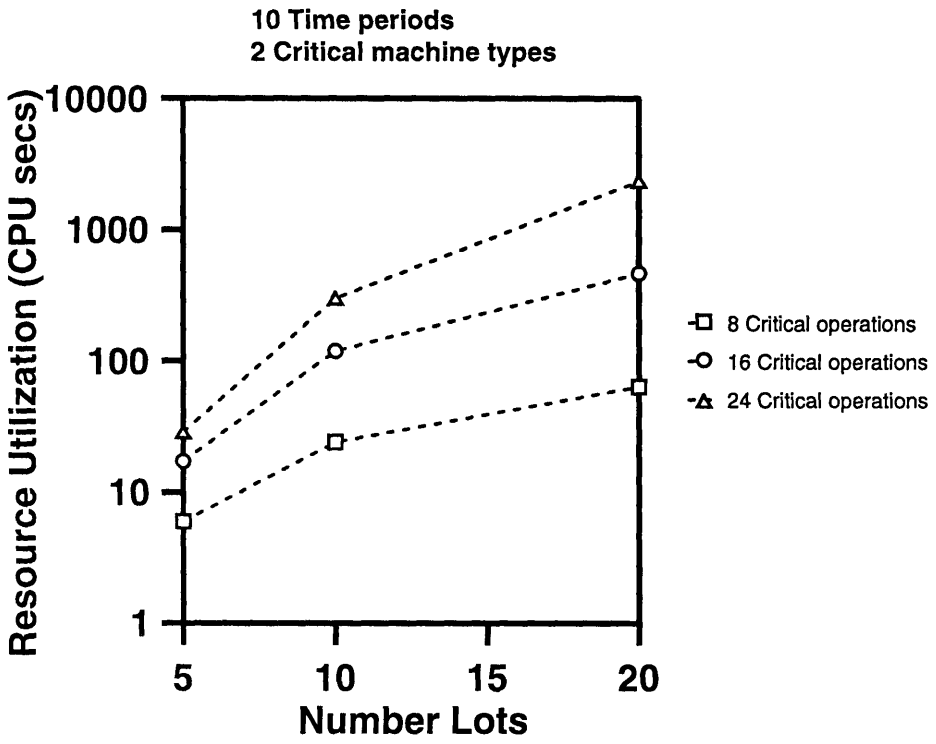


Figure 5-20: CPU usage as a Function of $L$ and $K$

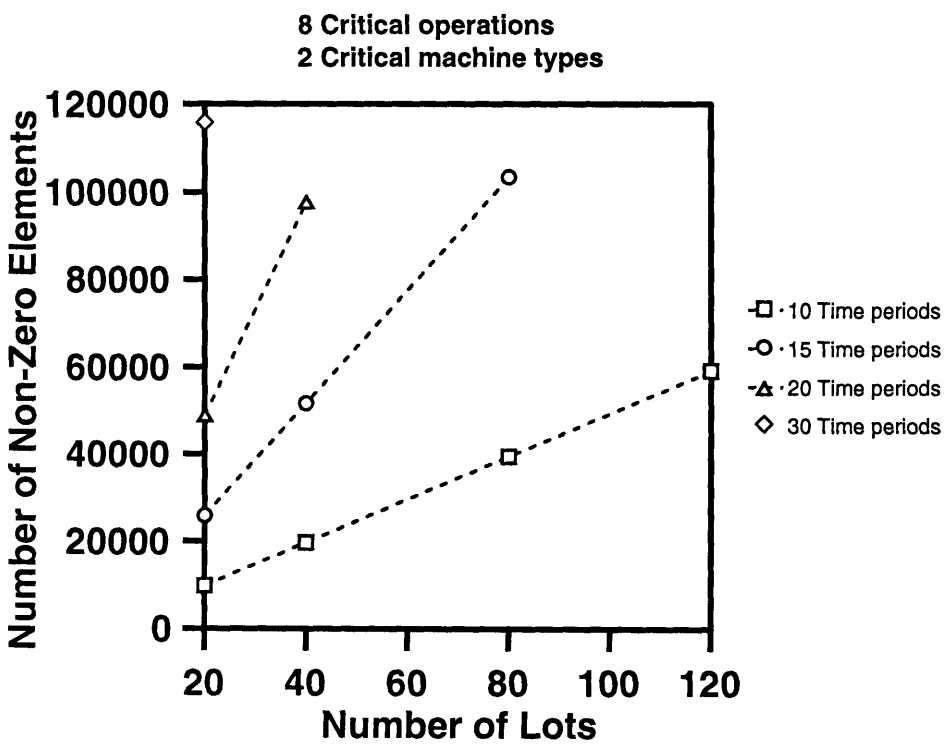


Figure 5-21: Number of Non-Zero Elements as a Function of $L$ and $I$

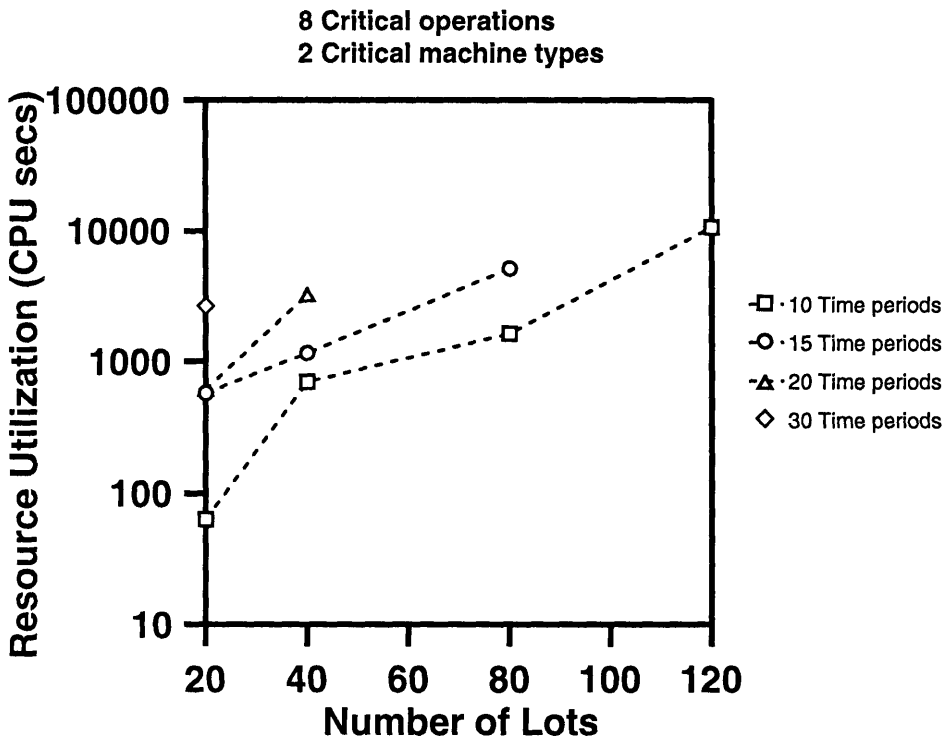


Figure 5-22: CPU usage as a Function of $L$ and $I$

# Chapter 6

# Conclusion

In this thesis, a model has been formulated to determine the feasibility of a release schedule. Various problem reduction techniques such as aggregation, selected focusing, and relaxation were developed and carried out to make the problem solvable. In addition, relaxing the problem resulted in a stochastic interpretation of the formulation. It was demonstrated that this quality diminished rapidly as the manufacturing system being analyzed operated closer to maximum capacity. In addition, MUAT was developed so that users of CAFE would be able analyze capacity issues using this model.

There is a plethora of additional research that can be done. As mentioned before, translating the results of the model into an operation loading schedule is a very difficult but worthwhile area to investigate. Developing more techniques to further simplify the model would prove invaluable in trying to solve very large problems. Furthermore, determining more necessary constraints would increase the chances that a feasible solution in the model is actually feasible in reality.

# Bibliography


[1] Yohanan Arzi and Adar Kalir. Estimating product workloads in a production system with alternative dissimilar machines operated in a produce-to-order environment. *Journal of Manufacturing Systems*, 65(4), 1994.

[2] *AT&T Technical Journal*, 65(4), Jul/Aug 1986. The entire issue is devoted to Manufacturing Technology.

[3] R. Bellman and R. Kalaba. *Dynamic Programming and Modern Control Theory.* Academic Press, 1965.

[4] Dimitri Bertsekas. *Nonlinear Programming- Lecture Notes.* MIT Graphic Arts, Cambridge, 1995.

[5] D.S Boning and M.B McIlrath. Guide to the process flow representation, vers. 3.1. *MIT MTL Memo 93-724*, September 1993.

[6] Anthony Brooke, David Kendrick, and Alexander Meeraus. *GAMS- A User's Guide.* The Scientific Press, San Francisco, first edition, 1988.

[7] Stanley B. Gershwin. Mit 2.852 spring term lecture (2/9/95).

[8] Stanley B. Gershwin. *Manufacturing Systems Engineering.* Prentice Hall, New York, 1994.

[9] James H. Greene. *Production and Inventory Control Handbook.* APICS. McGraw-Hill, New York, second edition, 1987.

[10] Peter Luh and Debra Hoitomt. Scheduling of manufacturing systems using the lagrangian relaxation technique. *IEEE Transactions on Automatic Control*, 38(7), July 1993.

[11] Michael B. McIlrath, Donald Troxel, Michael L. Heytens, Jr. Paul Penfield, Duane Boning, and Rajeev Jayavant. Cafe- the mit computer-aided fabrication environment. *IEEE Transactions on Components, Hybrids, and Manufacturing Technology*, Feb 1992.

[12] John Tsitsiklis. *Introduction to Mathematical Programming- Lecture Notes.* MIT Graphic Arts, Cambridge, 1995.

[13] Multi-criteria models for capacity analysis and aggregate planning in manufacturing systems. *International Journal of Production Economics*, 23(1–3), Oct 1991.

[14] Thomas E. Vollmann, William L. Berry, and D. Clay Whybark. *Manufacturing Planning and Control Systems.* Business One Irwin/APICS Series in Production Management. Business One Irwin, Homewood, Illinois, third edition, 1992.

[15] Wayne L. Winston. *Operations Research: Applications and Algorithms.* PWS-KENT, Boston, second edition, 1991.

[16] *UNIX Programmer's Manual Release 2.1(.7)*, May 1993. Found in Unix Man Pages.

# Appendix A

# Gamsin File

```
SETS
   I    All Time period    /
          T1
*          T1
         T2
*          T2
         T3
*          T3
         T4
*          T4
         T5
*          T5
         T6
*          T6
         T7
*          T7
         T8
*          T8
         T9
*          T9
         T10
*          T10
   /

   K    Operation order    / K1 * K12 /
* max K is the max number of operations of any lot
   L    Lot    /
         L1, L2, L3, L4, L5 /

   M    Machine Type    /
         M1
*          ("m1-a" "m1-b")
         M2
*          "m3"
   /

   B    Boulean    / B1 * B5 /
* max B is max lead time

   LK(L,K) Possible LK combinations /
          L1.(K1,K3,K5,K7,K9,
         K11,K6,K12)
          L2.(K1,K3,K5,K7,K9,
         K11,K6,K12)
          L3.(K4,K6,K8,K10,K2,K5,K7)
          L4.(K4,K6,K8,K10,K2,K5,K7)
          L5.(K4,K6,K8,K10,K2,K5,K7)
/

   LKRED(L,K) Possible LK combinations without the last op /
          L1.(K1,K3,K5,K7,K9,
         K11,K6)
          L2.(K1,K3,K5,K7,K9,
         K11,K6)
          L3.(K4,K6,K8,K2,K5,K7)
          L4.(K4,K6,K8,K2,K5,K7)
          L5.(K4,K6,K8,K2,K5,K7)
/
```

```
   LKENDS(L,K) Possible LK combinations for first and last operation only /
          L1.(K1,K12)
          L2.(K1,K12)
          L3.(K4,K10)
          L4.(K4,K10)
          L5.(K4,K10)
/

   LKFIRST(L,K) Possible LK combinations for first operation only /
          L1.K1
          L2.K1
          L3.K4
          L4.K4
          L5.K4
/

   LKLAST(L,K) Possible LK combinations for last operation only /
          L1.K12
          L2.K12
          L3.K10
          L4.K10
          L5.K10
/

   LKBRED(L,K,B) Possible LKB combinations without the last op / (
          L1.(K1,K3,K5,K7,K9,
         K11,K6)
          L2.(K1,K3,K5,K7,K9,
         K11,K6)
          L3.(K4,K6,K8,K2,K5,K7)
          L4.(K4,K6,K8,K2,K5,K7)
          L5.(K4,K6,K8,K2,K5,K7)
).(B1 * B5) /

   LKBPREC(L,K,B) Possible LKB combinations in Precedence constraint

   LKI(L,K,I) Possible LKI combinations / (
          L1.(K1,K3,K5,K7,K9,
         K11,K6,K12)
          L2.(K1,K3,K5,K7,K9,
         K11,K6,K12)
          L3.(K4,K6,K8,K10,K2,K5,K7)
          L4.(K4,K6,K8,K10,K2,K5,K7)
          L5.(K4,K6,K8,K10,K2,K5,K7)
).(T1 * T10) /

   LIB(L,I,B) Possible LIB combinations / (
L1, L2, L3, L4, L5).(T1 * T10).(B1 * B5) /

   LIBPREC(L,I,B)  Possible LIB combinations for prec constraint excluding D_l-1

   LKIVAR(L,K,I) Possible LKI for W variables

   LKMI(L,K,M,I) Possible LKMI combinations / (
         L1.((K1,K3,K5,K7,K9,
         K11).M1
         (K6,K12).M2)

         L2.((K1,K3,K5,K7,K9,
```

```
        K11).M1
        (K6,K12).M2)

         L3.((K4,K6,K8,K10).M1
        (K2,K5,K7).M2)

         L4.((K4,K6,K8,K10).M1
        (K2,K5,K7).M2)

         L5.((K4,K6,K8,K10).M1
        (K2,K5,K7).M2)

        ).(T1 * T10) /
   LKMIVAR(L,K,M,I) Possible LKMI for W variables

;

************************* Model Raw Data **************************

PARAMETERS
      S(L) Lot start date /

   L1 1
   L2 2
   L3 3
   L4 4
   L5 5 /

      D(L) Lot due date /

   L1 6
   L2 7
   L3 8
   L4 9
   L5 10 /

   LEAD(L) Lot lead time

   STEPINC(L,K)  Increment between steps /
   L1.K1 2
   L1.K3 2
   L1.K5 1
   L1.K7 2
   L1.K9 2
   L1.K11 1
   L1.K6 1


   L2.K1 2
   L2.K3 2
   L2.K5 1
   L2.K7 2
   L2.K9 2
   L2.K11 1
   L2.K6 1


   L3.K4 1
   L3.K6 1
```

```
   L3.K8 2

   L3.K2 2
   L3.K5 1
   L3.K7 1

   L4.K4 1
   L4.K6 1
   L4.K8 2

   L4.K2 2
   L4.K5 1
   L4.K7 1

   L5.K4 1
   L5.K6 1
   L5.K8 2

   L5.K2 2
   L5.K5 1
   L5.K7 1

/

TIMEPER(I) Time period as a variable instead of an index
;

SCALAR
   TFLENGTH Time Frame Length /10/
   LOWDIF  Difference between first day of time frame and of all time /0/
;

********************** Equations to process raw data *******************

LEAD(L) = D(L) - S(L) ;

LIBPREC(L,I,B) = LIB(L,I,B)$((((ORD(I)-LOWDIF) GE S(L)) AND

          ((ORD(I)-LOWDIF) LE (ORD(B)+S(L)-1))) AND

          ((ORD(I)-LOWDIF) LE (D(L)-2))) ;

LKBPREC(L,K,B) = LKBRED(L,K,B)$(ORD(B) LE LEAD(L))  ;

LKIVAR(L,K,I)$LKI(L,K,I) = YES$(((ORD(I)-LOWDIF) GE S(L)) AND
                ((ORD(I)-LOWDIF) LE D(L)));

LKMIVAR(L,K,M,I)$LKMI(L,K,M,I) = YES$(((ORD(I)-LOWDIF) GE S(L)) AND
((ORD(I)-LOWDIF) LE D(L)))  ;

TIMEPER(I) = ORD(I) - LOWDIF;

************** DATA USED DIRECTLY BY LP******************

PARAMETER
   TAU(L,K) Average operation time (in hours) /

   L1.K1 10.000000
   L1.K3 10.000000
```

```
L1.K5 10.000000
L1.K7 10.000000
L1.K9 10.000000
L1.K11 10.000000
L1.K6 20.000000
L1.K12 20.000000

L2.K1 10.000000
L2.K3 10.000000
L2.K5 10.000000
L2.K7 10.000000
L2.K9 10.000000
L2.K11 10.000000
L2.K6 20.000000
L2.K12 20.000000

L3.K4 10.000000
L3.K6 10.000000
L3.K8 10.000000
L3.K10 10.000000
L3.K2 20.000000
L3.K5 20.000000
L3.K7 20.000000

L4.K4 10.000000
L4.K6 10.000000
L4.K8 10.000000
L4.K10 10.000000
L4.K2 20.000000
L4.K5 20.000000
L4.K7 20.000000

L5.K4 10.000000
L5.K6 10.000000
L5.K8 10.000000
L5.K10 10.000000
L5.K2 20.000000
L5.K5 20.000000
L5.K7 20.000000

/

    TIMEDIF(L,K) Time difference between k and k+1 crit op /

L1.K1 0.500000
L1.K3 0.500000
L1.K5 0.333333
L1.K7 0.500000
L1.K9 0.500000
L1.K11 0.333333
L1.K6 0.666667


L2.K1 0.500000
L2.K3 0.500000
L2.K5 0.333333
L2.K7 0.500000
L2.K9 0.500000
L2.K11 0.333333
```

```
  L2.K6 0.666667


  L3.K4 0.333333
  L3.K6 0.333333
  L3.K8 0.500000

  L3.K2 0.833333
  L3.K5 0.666667
  L3.K7 0.666667

  L4.K4 0.333333
  L4.K6 0.333333
  L4.K8 0.500000

  L4.K2 0.833333
  L4.K5 0.666667
  L4.K7 0.666667

  L5.K4 0.333333
  L5.K6 0.333333
  L5.K8 0.500000

  L5.K2 0.833333
  L5.K5 0.666667
  L5.K7 0.666667

/

    TIMEDIFBEG(L,K) Time difference between start date and first crit op /

  L1.K1 0.000000
  L2.K1 0.000000
  L3.K4 0.166667
  L4.K4 0.166667
  L5.K4 0.166667
/

    TIMEDIFEND(L,K) Time difference between due date and last crit op /

  L1.K12 0.666667
  L2.K12 0.666667
  L3.K10 0.500000
  L4.K10 0.500000
  L5.K10 0.500000
/

    ME(M) Machine efficiency /

  M1 1.000000
  M2 1.000000 /

    E(M) Number of machines /

  M1 2
  M2 1 /
```

```
;

SCALARS

   KAP Fraction of resource used per unknown lots /0.000000/
   N    Number of hours in a time period /30/
   U    Utilization factor   /1.000000/

;

VARIABLES
W(L,K,I)  Fraction of an operation
ET(L,K)   Expected time period operation occurs
Z         Total load cost;

POSITIVE VARIABLES    W;

EQUATIONS
LOAD          Total load
UNITY(L,K)    Unity constraint
CAPACITY(M,I)  Capacity constraint
EXPTIME(L,K)  Expected time period constraint
EXPTIMEBEG(L,K) Expected time period constraint for first op
EXPTIMEEND(L,K) Expected time period constraint for last op
ETEQN(L,K)    Calculates Expected time
TIMEPERIOD(L,I) Time Period Constraint
PRECEDENCE(L,K,B)  Precedence constraint ;

LOAD .. Z =E=  SUM((L,K)$LKLAST(L,K), ET(L,K)) ;
UNITY(L,K) $LK(L,K)  .. SUM(I$LKIVAR(L,K,I),W(L,K,I)) =E= 1 ;
CAPACITY(M,I)  .. SUM((L,K),TAU(L,K)*W(L,K,I)$LKMIVAR(L,K,M,I)) =L= N*ME(M)*U*E(M) ;
EXPTIME(L,K) $LKRED(L,K) ..
 SUM(I$LKIVAR(L,K,I),TIMEPER(I)*W(L,K+STEPINC(L,K),I)) =G=
 SUM(I$LKIVAR(L,K,I),TIMEPER(I)*W(L,K,I))  + TIMEDIF(L,K) ;
EXPTIMEBEG(L,K) $LKFIRST(L,K) ..
 SUM(I$LKIVAR(L,K,I),TIMEPER(I)*W(L,K,I)) =G= S(L) + TIMEDIFBEG(L,K) ;
EXPTIMEEND(L,K) $LKLAST(L,K) ..
 SUM(I$LKIVAR(L,K,I),TIMEPER(I)*W(L,K,I)) =L= D(L) + 1 - TIMEDIFEND(L,K) ;
ETEQN(L,K) $LK(L,K) .. ET(L,K) =E=
 SUM(I$LKIVAR(L,K,I),TIMEPER(I)*W(L,K,I)) ;
TIMEPERIOD(L,I) ..
 SUM(K,TAU(L,K)*W(L,K,I)$LKIVAR(L,K,I)) =L= N ;
PRECEDENCE(L,K,B) $LKBPREC(L,K,B)  ..
   SUM(I$LIBPREC(L,I,B),W(L,K+STEPINC(L,K),I))  =L=
    SUM(I$LIBPREC(L,I,B),W(L,K,I)) ;

OPTION LIMROW = 1000 ;

OPTION LP=MINOS5;

OPTION ITERLIM = 100000;

OPTION RESLIM = 20000;

MODEL  TOY    Machine Utilization Toy Problem / ALL /  ;

SOLVE  TOY    USING LP MINIMIZING Z ;

OPTION CAPACITY:4:0:2; DISPLAY "Maximum Capacity during time period ",CAPACITY.UP;
```

```
OPTION CAPACITY:4:0:2; DISPLAY "Expected load during time period ",CAPACITY.L;

OPTION W:4:0:3; DISPLAY "Time Periods",W.L;

OPTION ET:4:0:3; DISPLAY "Expected time perdiod operation occurs",ET.L;
```