11-1993

A Robust Aggregation Approach To Simplification Of Manufacturing Flow Line Models

Paul Savory
University of Nebraska at Lincoln, psavory2@gmail.com

A ROBUST AGGREGATION APPROACH TO SIMPLIFICATION OF MANUFACTURING FLOW LINE MODELS

by

Paul Andrew Savory

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

ARIZONA STATE UNIVERSITY

December 1993

A ROBUST AGGREGATION APPROACH TO SIMPLIFICATION

OF MANUFACTURING FLOW LINE MODELS

by

Paul Andrew Savory

has been approved

November 1993

APPROVED:

_____,Chairperson

_____

_____

Supervisory Committee

ACCEPTED:

_____

Department Chairperson

_____

Dean, Graduate College

ABSTRACT

One of the more difficult tasks facing a modeler in developing a simulation model of a discrete part manufacturing system is deciding at what level of abstraction to represent the resources of the system. For example, questions about plant capacity can be modeled with a simple model, whereas questions regarding the efficiency of different part scheduling rules can only be answered with a more detailed model. In developing a simulation model, most of the actual features of the system under study must be ignored and an abstraction must be developed. If done correctly, this idealization provides a useful approximation of the real system. Unfortunately, many individuals claim that the process of building a simulation model is an "intuitive art." The objective of this research is to introduce aspects of "science" to model development by defining quantitative techniques for developing an aggregate simulation model for estimating part cycle time of a manufacturing flow line. The methodology integrates aspects of queueing theory, a recursive algorithm, and simulation to develop the specifications necessary for combining resources of a flow line into a reduced set of aggregation resources. Experimentation shows that developing a simulation model with the aggregation resources results in accurate interval estimates of the average part cycle time.

ACKNOWLEDGMENTS

I wish to express my gratitude to my major professor, Dr. Gerald Mackulak, for his guidance through the process of obtaining a doctorate. I would also like to thank Dr. Jeffery Cochran for all his assistance. I have worked with both of these individuals through the classroom, on Systems Simulation Laboratory project work and joint publications and through these esperiences I have learned not only how to perform research, but also how to work on and run a research project. The skills and tricks they have taught me will aid me throughout my career and I am thankful I had the opportunity to learn for two of the best.

In addition, I wish to thank Dr. Jeffrey Arthur of Oregon State University for his continued support throughout my education. I also appreciate the assistance of Dr. Douglas Montgomery and Dr. Bert Keats who have allows allowed me to stop by their offices with questions about my research and my teaching.

TABLE OF CONTENTS

APPENDIX Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

> If you want to understand some aspect of the Universe, it helps if you simplify it as much as possible and include only those properties and characteristics that are essential to understanding. If you want to determine how an object drops, you don't concern yourself with whether it is new or old, is red of green, or has on odor or not. You eliminate those things and thus do not needlessly complicate matters. The simplification you call a model or a simulation and you can present it either as an actual representation on a computer screen or as a mathematical relationship. ... Such simplified simulation make it far easier to grasp a phenomenon than it would be if we had to study the phenomenon itself.
>
> Hari Seldon, mathematician
> (Asimov, 1988, p. 138)

1.1 Research Problem Statement

In developing a simulation model of a discrete part manufacturing system, a modeler must decide at what level of abstraction to represent the resources of the system. For example, questions about plant capacity can be modeled with a simple model, whereas questions regarding the efficiency of different part scheduling rules can only be answered with a more detailed model (Thesen and Travis, 1988). Unfortunately, many claim that the process of building a simulation model is an "intuitive art" (Emshoff and Sissin, 1970; Shannon, 1975; MacNair and Sauer, 1985; and Pritsker, 1986 are a small subset). The objective of this research is to introduce aspects of "science" to model development by defining a quantitative methodology for developing an aggregate simulation model of a manufacturing flow line system.

1.2 Problem Environment

Computer simulation is the process of developing a mathematical-logical model of a real system and experimenting with this model on a computer (Pritsker, 1986). It is one of the most important operations research techniques (Lane, Mansour, and Harpell, 1993). It's many uses range from comparing alternative systems to answering capacity and feasibility questions. Simulation modeling has its roots in computer science, mathematics, and statistics (Murray and Sheppard, 1987). To use simulation correctly and intelligently, the practitioner is required to have training in each of these different fields. Shannon *et al.* (1985) estimate that a simulation practitioner must have about 720 hours of formal classroom instruction plus another 1440 hours of outside study to gain this basic knowledge. This estimate emphasizes one of the significant disadvantages of simulation: the quality of the analysis depends on the quality of the model.

One of the more difficult tasks of model building is determining at what level of abstraction to model the resources of the system. According to Shannon (1975), "Model building requires an ability to analyze a problem, abstract from it its essential features, select and modify basic assumptions that characterize the system, and then enrich and elaborate the model until a useful approximation results." He concludes by stating that the successful approach to model building appears to proceed on the basis of elaboration and enrichment. One starts with a very simple model and elaborates it until it clearly represents the system. Pegden *et al.* (1990) agrees and remarks that, "this process of system abstraction and simplification is the essence of modeling art."

In contrast, McHaney (1991) proposes that simulation is neither an art nor a science, but both. He asserts that the creativity and instincts used are akin to an art, while the methodology involved in model creation and analysis are based on computer science and mathematical principles. Therefore, elements of both art and science exist in modeling (Figure 1.1).

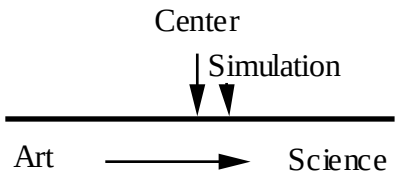


Figure 1.1. Art versus Science: where simulation falls [from McHanely, 1991].

Since a model is a description of a system, it is also an abstraction or simplification of that system (Pritsker, 1986). Most problem situations are enormously complex, containing an almost infinite number of elements, variables, parameters, relationships, constraints, etc. (Shannon, 1975). When building a model, an infinite number of facts can be included followed by an endless amount of time gathering detailed facts about any situation and defining the relationships among them. Shannon (1975) describes the following situation:

> Consider the simple act of taking a piece of paper and writing a letter. One could study the detailed chemical composition of the paper, lead, and erasure; the effect of atmospheric conditions on the moisture content of the paper and its effect on the friction of the pencil lead as it moves across the paper; the statistical distribution of the letters in the sentences; etc.

If the only aspect of interest is whether a letter is sent or not, then none of these details are pertinent. In developing a simulation model, most of the actual features of the system under study must be ignored and an abstraction must be developed.

The representation or model of a system is not unique. Depending upon the objective of the study, the same system can be represented in a variety of ways giving different types and amounts of information (Neelamkavil, 1987). The level of detail in a model is usually determined by the specific objective of the modeling effort (Emshoff and Sisson, 1970). For example, Thesen and Travis (1988) remark that questions about plant

capacity can be modeled with a simple model, while questions regarding the efficiency of different part scheduling rules can only be answered with a more detailed model.

Modeling is a balancing act (Balci, 1989). On one hand, a model should include the essential elements of the system, and on the other hand, it should not include unnecessary detail. Missing an essential element may invalidate the representation provided by the model. Inclusion of unnecessary detail only makes the model unnecessarily complex and difficult to analyze. The natural tendency of the novice is to include too much detail, while the more experienced modeler tends toward greater abstraction (Sadowski, 1989). Advantages for developing an abstract simulation model include (Gordon, 1969; Pedgen *et al.,* 1991):

- a reduced run length,
- a less complex model,
- easier to animate,
- easier to debug, validate, modify and document,
- less demand of programming resources (queues and systems variables),
- less data dependent answers.

An argument often used for the complex model is the need for precise results. Where this argument fails is that the increased run-time for the more complex model may lead to fewer replications, which in turn produces wider confidence intervals on the performance measure (Pegden *et al.,* 1990). Hence, the more detailed model may actually produce less precise results. In addition, consider the case of a manufacturing line where all operations are specified as data distributions. For example, ten production steps requires ten processing times, one for each step. If these estimates are not accurate, it would be better to combine or aggregate several or all of the steps and guess a combined distribution. This will reduce the variability of the estimate, since as the data is aggregated the importance of each individual time is minimized. A final disadvantage of

a large and complex model is it is more likely to contain undetected bugs that can introduce errors of a much larger magnitude than a simpler model.

1.3 Scope and Objectives

In developing a simulation model, most of the actual features of the system under study must be ignored and an abstraction must be developed. If done correctly, this idealization provides a useful approximation of the real system, or at least certain parts of the real system. Abstracting a system into a simulation model involves three techniques: simplification, aggregation, and substitution.

The easiest method, simplification, is the omission of certain details from the model, such as infrequent machine breakdowns, small travel times, etc. (Pegden *et al.*, 1990). This simplification process entails stripping away unimportant details or the assumption between relationships (Shannon, 1975). This approach assumes that not all factors are equally important in determining system behavior. The task for the modeler is determining which factors are critical, and which are not.

The second method involves aggregating or lumping details into a single, approximately equivalent function (Pegden *et al.,* 1990). Pedgen *et al.* (1990) discusses the example of a manufacturing flow line in which an operator may perform several distinct tasks on a part as it moves through a workstation. Rather than individually modeling each of these tasks, one can model the entire operation as a single process. For example, the load time for a part on the machine might be aggregated with the machine process time to have a single part time on the machine.

A third technique in developing an abstract model involves substituting a simpler but approximate process for a more complex one. For example, consider a manufacturing cell with several different lathes, each with slightly different performance characteristics. These can be approximated by a set of parallel, identical lathes. Another common

substitution involves replacing a stochastic process with a constant process (replacing a probabilistic service time with a deterministic service time). For instance, it may be known that the process time for a part follows a normal distribution with a very small standard deviation. This could be simplified by modeling the processing time as a constant time based on the mean.

Facing competitive pressure from both domestic and foreign sources, today's manufactures must strive to keep delivery promises and reduce inventory (Cheng, 1990). Success depends upon a manufacture's ability to reliably predict part cycle times. The ability of a manufacture to predict part cycle times accurately in its production system has two important implications (Cheng, 1990): (1) assignment of reliable and attainable due-dates to job orders, and (2) accurate assessment of the work-in-process inventory in the production system. Not only do both of these strive to win customer satisfaction by promising and delivery on specified dates but they also allow for a reduction in costs.

The objective of this research is to develop a formal methodology for creating an aggregate simulation model that can be used to estimate part cycle time. The scope of this research will be limited to flow line manufacturing systems. The methodology operates by aggregating or lumping together resources of the system to develop the specification for an aggregation simulation model that accurately estimates the part cycle time.

1.4 Research Assumptions

This research will proceed based on the following basic assumptions:

(1) The decision process for aggregating simulation resources will be studied from a *predictive* point of view. Such an approach focuses on identifying the impact of key or significant resources of a system. In comparison, the *reactive* point of view requires the generation of the complete model (Figure 1-2). It explores the

aggregation issue by running the full or complete model, and based on the output, decides an appropriate procedure for aggregating system resources. The noticeable discrepancy is that if the full model needs to be run, why study developing a reduced or aggregate model? It is the premise of this research to aid a modeler in developing the initial simulation model.

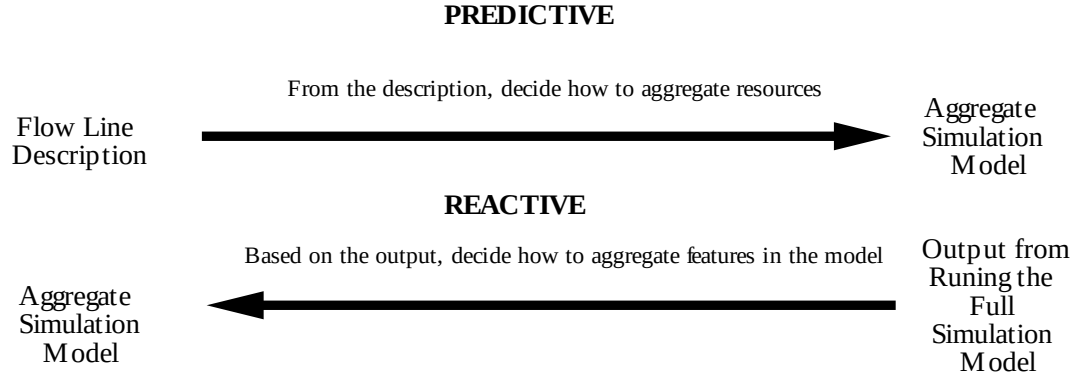


Figure 1.2. Comparison of two approaches for aggregating resources in a simulation model.

(2) The manufacturing system that this research explores is a production flow line (flow shop) system. Systems of this type are widely used in industry to represent situations in which parts (or customers, or telephone calls) arrive to a service area, obtain the service they require, and then move on to the next service area or leave the system. Systems of this type are a subset of the broad class of discrete part manufacturing systems. It is hypothesized that future research will be able to extend the research methodology to other discrete manufacturing systems (manufacturing cell, job shop, flexible manufacturing system, etc.). Discussion on continuous or hybrid systems will be excluded.

(3) All shop floor data is readily available. In theory, this information is available on process flow diagrams, but in practice it is not always known. It can be assumed that in a fully-integrated, computerized factory (with data checking) of the future,

this information can be obtained. In addition, common sense inherently dictates that if a part flow is currently in operation or is being modeled for such, the information about the operational steps required to make a part must be known.

(4) The success of the aggregation procedure will be judged on how well it estimates a single performance variable, namely the average cycle time (sojourn time) of a part to wait and be serviced by all stations (resources) of the flow line. In addition to this variable being important for planning delivery dates, it is also useful for reducing costs. Mott and Tumary (1992) discuss that since material carrying costs are directly related to the value of the assets (parts being produced) and the length of time those assets remain in the process, estimation of the average cycle time of a part is an important value for system evaluation.

(5) Application of the research methodology requires that the underlying manufacturing system be stable and operating at less than capacity. Specifically, the arrival rate of a part to the flow line and the resource service times are such that the associated queue of the resource is not growing unbounded. This can be tested for all flow line stations by checking that the mean arrival rate divided by the service rate multiplied by the number of servers is less than one.

(6) The relationship between the different production stations or resources in the flow line is independent for estimating the cycle time of a part. That is, it will be assumed that the order of resources has little if any impact on estimating the cycle time of a part through a flow line. The case when independence is not assumed and the order of resources in the flow line is important and will be discussed in Chapter 4.

(7) A final constraint on the system is that there are no feedback loops in the production operation.

1.5 Major Research Tasks

A high level view of this research is illustrated in Figure 1.3. Applying the aggregation methodology to a production flow manufacturing system description results in the specifications of the equivalent aggregate system.

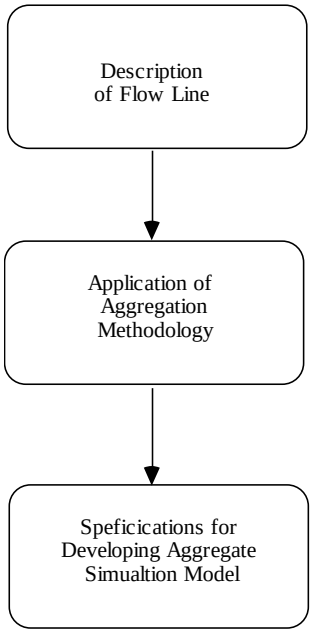


Figure 1.3. The research methodology decision process.

The major tasks identified for achieving the objectives of this research are summarized as follows:

(1) Development of a formalism to describe a production flow manufacturing system. This description provides a necessary foundation for identifying and collecting information that the aggregation methodology requires.

(2) Creation of a formalism to describe the aggregate system. It summarizes all the information necessary for the modeler to develop an aggregate simulation model of the system.

(3) Identification of a procedure for computing the average cycle time of an aggregation resource.

(4) Development of techniques to estimate the mean service time of an aggregate resource.

(5) Origination of a method for describing the mean service time of an aggregation resource in terms of its aggregated resource service means.

(6) Identification of a method for creating the aggregate simulation model using the original resource service time distributions.

A more detailed description of these tasks and their relationship is presented in Chapter 3, RESEARCH METHODOLOGY.

1.6 Original Contribution of the Research

The objective of this research is to create a formalized methodology for developing an aggregate simulation model of a production flow line which estimates part cycle time. Upon completion of this research and meeting of the research objectives, this research has the following original contributions:

(1) Development of a methodology for aggregating resources in a simulation model of a production flow line.

(2) Development of formalisms for describing a production flow line and its aggregate equivalent. These formalisms provide a foundation from which a flow line can be defined and compared.

(3) Identification of a method for determining the service mean of an aggregate resource. This involves applying queueing formulas backwards, in that the mean service time of an aggregation resource is estimated from the average cycle (waiting) time. This differs from the more common approach of specifying the mean service time and computing the cycle (waiting) time.

(4) Creation of a technique for weighting the resources of an aggregate resource so as to estimate the aggregate resources service time mean. This is accomplished through the use of a recursive algorithm.

(5) Demonstration that analytical techniques such as queueing analysis can be integrated with simulation to reduce the effort necessary to address questions that simulation can answer.

1.7 Organization of the Dissertation

This dissertation consists of five chapters. Chapter 1 presents the research problem statement, defines the problem environment, identifies the scope and objectives of this research, states the research assumptions, defines the major research tasks, and lists the original contribution of this research. Chapter 2 reviews past research in the areas covered by this research. It also introduces techniques used by the aggregation methodology and thus provides a foundation from which the research methodology is developed.

The research methodology is presented in Chapter 3. The chapter begins by presenting an overview of all the major research tasks and the links between them. This is accomplished by presenting the specific steps of the research methodology. The remainder of the chapter expands and discusses the details involved with each of the steps. The analysis of an example flow line is continued throughout the chapter to illustrate each step of the aggregation methodology.

Chapter 4 is allocated for illustrating the application of the aggregation methodology developed in the previous chapter. It discusses the impact of applying the aggregation methodology by studying its application for three types of flow line system: a flow line with all exponential servers, a flow line with only single capacity servers, and a flow line with any number of servers of any capacity. For each of these different cases,

the results of applying the methodology is explored. In addition, a computer program which implements the aggregation methodology is discussed. Chapter 5 provides a summary of the research results, summarizes the limitations of the research, and offers recommendations for future research work on this topic.

The reference section lists all literature reviewed in regard to this research. The format follows that of the journal *Simulation* published by the Society for Computer Simulation.

CHAPTER 2

LITERATURE SURVEY

This chapter identifies the important research concepts necessary for creating the aggregation methodology for estimating the cycle time of a part. Section 2.1 defines computer simulation by discussing the simulation model development process. Section 2.2 discusses the components that make up a discrete manufacturing systems. It also lists the different types of flow line manufacturing systems and reviews commonly studied performance measures. Section 2.3 explores the use of queueing analysis for estimating the steady state performance of a queueing system. This section concludes by discussing previous research of tandem queue (flow line) system. With no previous formal research for developing an aggregate simulation model, Section 2.4 reviews the general concept of aggregation and reviews how aggregation is performed in other areas. Section 2.5 summarizes the major concepts presented in Chapter 2.

2.1 Introduction to Simulation

Simulation is one of the most important operations research techniques (Lane, Mansour, and Harper, 1993). Its uses range from answering holding capacity and production feasibility questions to comparing alternative system routing and scheduling plans. This section provides an introduction to developing a simulation model. Section 2.1.1 introduces the concept of a model. Section 2.1.2 explores computer simulation models, a specific type of model. Section 2.1.3 reviews the procedures for developing a simulation model. Section 2.1.4 summarizes procedures for generating random variables.

2.1.1 What is a Model?

A model is a simplified representation of a system (or process or theory) intended to enhance our ability to understand, predict, and possibly control the behavior of the

system (Neelamkavil, 1987). More formally, Shannon (1975) defines a model to be “a representation of an object, system, or idea in some form other that of the entity itself “ (Shannon, 1975). Pritsker (1986) expands on this definition and states that models are descriptions of systems. A system is a collection of regularly interacting or interdependent components (such as machines, people, information, and communications), acting as a unit in carrying out an implicitly or explicitly defined mission (Maisel and Gnugnoli, 1972).

Modeling is not new; mankind has been conceptualizing and developing models since he began to understand and manipulate his environment (Shannon, 1975). The concept of representing some objects, system, or idea with a model, is so general that it is difficult to classify all the functions models fulfill. Elmaghraby (1968) identifies at least five common uses for models:

- an aid to thought
- an aid to communication
- an aid to training and instruction
- a tool of prediction
- an aid to experimentation

Shannon (1975) remarks that models may either be *descriptive* or *prescriptive*. A descriptive model is useful for explaining and/or understanding while a prescriptive model predicts and/or duplicates a systems behavior characteristics.

There are many methods to classify models, unfortunately none is completely satisfactory, although each serves a particular purpose. Some of these classification schemes are as follows:

- static versus dynamic
- deterministic versus stochastic
- discrete versus continuous

• iconic versus analog versus symbolic

Gordon (1969) classifies models based on whether they are physical or mathematical (Figure 2.1). A model is said to be a physical model whenever the modeling representation is physical and tangible, with model elements made of material and hardware (Jacoby and Kowalik, 1980). Correspondingly, a model is said to be a mathematical or formal model when a set of mathematical or logical relations are used to describe a system. A second distinction is between *static* models and *dynamic* models. In the case of mathematical models, a third distinction is the technique employed in solving the model. Specifically, a distinction is made between *analytic* and *numeric* methods.

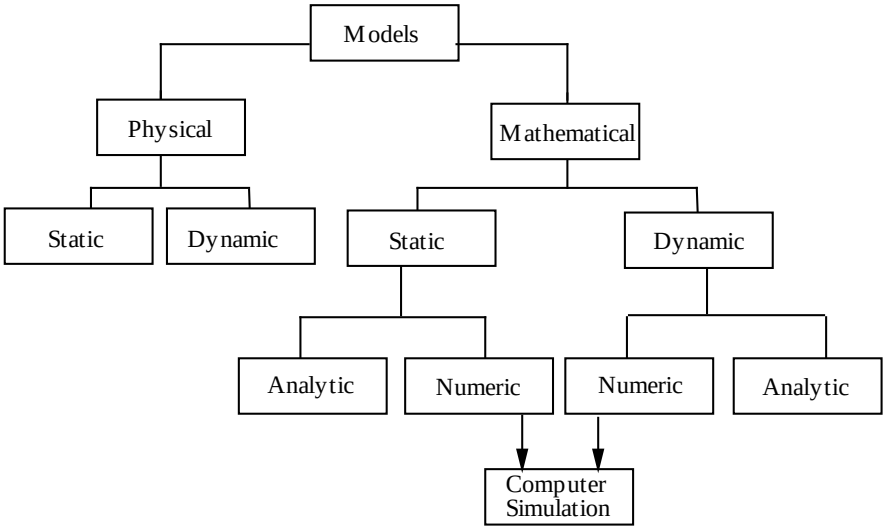


Figure 2.1. Classification of models [modified from Gordon, 1969].

Shannon (1975) uses a different approach for describing models. He describes models on a continuous spectrum, starting with exact models or prototypes of reality and proceeding to completely abstract mathematical models (Figure 2.2).

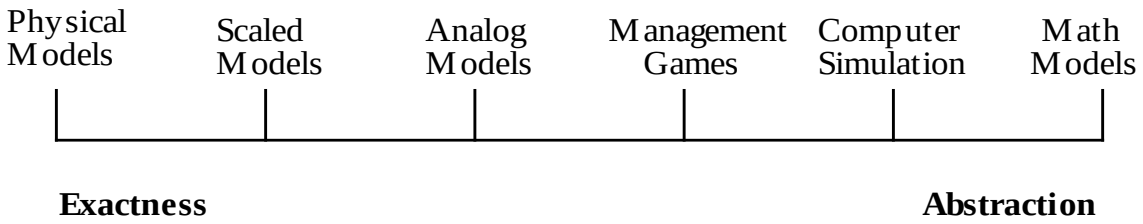


Figure 2.2. Spectrum classification of models [from Shannon, 1975].

A model goes through a set of development steps. Pritsker (1986) shows a pictorial view of the model building approach (Figure 2.3). He states that "a system is a collection of items from a circumscribed sector of reality that is the object of study or interest." The first step in building a model is developing the purpose for modeling. Based on this purpose, the boundaries of the system and a level of modeling detail are established. The desired performance measures and design alternatives are also included in the model. Assessment of design alternatives in terms of the specified performance measures are considered as model outputs. Once the development process is complete, the conceptual model is ready for implementation.

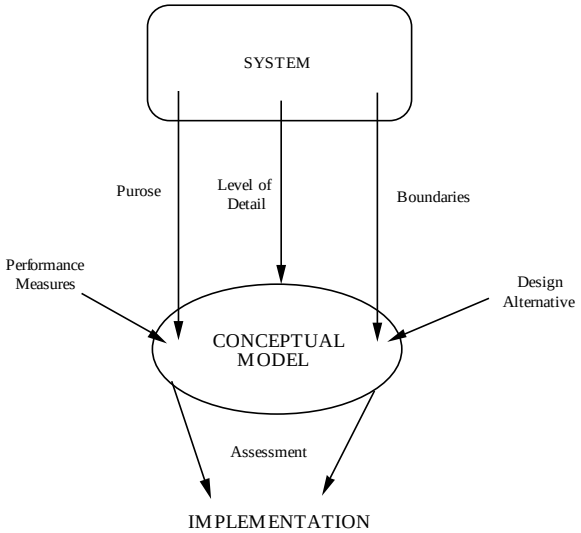


Figure 2.3. Factors affecting the development of a model [from Pritsker, 1986].

2.1.2 Computer Simulation

Naylor (1979) defines simulation as "... a numerical technique for conducting experiments on a digital computer, which involves logical and mathematical relationships that interact to describe the behavior and structure of a complex real-world system over extended periods of time." Computer simulation is therefore an expensive and complicated process and should be used only under the following circumstances (McHaney, 1991):

- The real system does not exist and it is too costly, time-consuming, hazardous, or impossible to build a prototype.
- The real system exists but experimentation is expensive, hazardous, or seriously disruptive.
- Mathematical modeling of the system has no practical analytical or numeric solutions.

Simulation is not without its drawbacks (Law and Kelton 1982):

- Simulation models are often expensive and time consuming to develop.
- Simulation models give estimates of performance characteristics, and offers few procedures for optimization.
- Simulation models produce large quantities of output which possible can give a false impression that the model is valid, when in fact this cannot be determined until the output is studied.

A simulation model is made up of many parts. Shannon (1975) states that almost every simulation model consists of some combination of the following ingredients:

- *components* - the parts when taken together make up the system under study.
- *parameters* - the input or specific characteristics which describe the system.
- *variables* - the output responses from running the simulation model. Two type of variables in a model of a system: *exogenous* and *endogenous*. Exogenous

variables are the input variables, those that originate or are produced outside of the system. They are independent of running the simulation model. Endogenous variables are those produced within the system or result from external causes. The can be thought of as the dependent variables.

- *functional relationships* - describes the relationships between variables and parameters in a system. These relationships are either deterministic or stochastic in nature.
- *constraints* - are limitations imposed on the values of the variables or on how resources can be allocated or expanded. An example of a constraint is a requirement that an automatic guided vehicle can only transport one part at any given time.
- *criterion function* - an explicit statement of the objectives or goals of the system and how they are to be evaluated.

2.1.3 Developing a Simulation Model

As with all models, the purpose of a computer simulation model is to provide a representations of a real system (Anderson *et al.,* 1988). Developing this model is a highly complex task which rarely is aided by the assistance of a computer. Table 2.1 presents a list of published articles which describe the simulation process, diagnose the common problems and pitfalls that can occur during a simulation study, or offer advise on how to perform a study. Many excellent textbooks have also been written which describe the simulation modeling process (Pegden *et al.,* 1990; Pritsker, 1986; Shannon, 1975; Banks and Carson, 1984 are a small subset).

Table 2.1. Published papers describing the simulation process.

Author	Subject
Annino and Russell (1979)	Reviews the ten most common causes of simulation failure and outlines the procedures for overcoming them.
Dietz (1992)	Outlines the format of a successful simulation project.
Fossett, Harrison and Weintrobs (1991)	Develops and tests a procedure for evaluating a simulation model.
Law (1986)	Provides an overview of how simulation is used to model manufacturing systems.
Law and McComas (1986)	Discusses the ten potential pitfalls in the areas of model development, simulation software, modeling randomness, and the design and analysis of simulation experiments.
Mott and Tumay (1992)	Presents a strategy for showing how a simulation study can be justified.
Nance (1983)	Summarizes current procedure of developing a simulation model.
Osborne and Watts (1977)	Summarizes the general principles of model classification, construction, and validation.
Pollacia (1989)	Introduces to the main concepts of discrete simulation.
Sadowski (1989)	Presents an approach for conducting a simulation project that will aid in avoiding many common problems and pitfalls.
Schoemaker (1978)	Reviews the “art” of simulation and discuss the stumbling blocks that may occur in a study.
Schruben (1983)	Introduces discrete event modeling by discussing the important issues related to model development.
Thesen and Travis (1988)	Introduces the uses of simulation, the underlying concepts, and the types of computer packages available.

Ulgen (1991)	Reviews proper management techniques necessary for making a successful simulation project.
Weiner, Grant, and Coffman (1986)	Suggests methods for establishing the credibility of industrial manufacturing simulation studies.

Figure 2.4 (Mackulak *et al.*, 1987) presents a high level flowchart of the steps required in a simulation study. The simulation process can be broken into the following stages or tasks (a modification of Pegden, *et al.*, 1990; Pritsker, 1986 and Shannon, 1975 present similar lists):

(1) Problem Definition - clearly identify the goals of the study to identify its purpose. The first task in a simulation project is the construction of a clear definition of the problem and an explicit statement of the objectives or goals of the analysis (Pritsker, 1986; Shannon, 1975; Emshoff and Sisson, 1970). Both Pritsker (1986) and Shannon (1975) agree that the formulation of a problem is a continuing process throughout the simulation study.

(2) Project Planning - decide on the personnel, management support, computer hardware, and software resources to perform the study.

(3) System Definition - determine the boundaries and restrictions to be used in defining the system (or process) and investigating how the system works. Once an initial problem statement and all goals are identified, the task of formulating a model beings (Pritsker, 1986). Both static and dynamic elements must be identified. The static description defines the elements of the system and the characteristics of those elements. The dynamic description defines the way in which elements in the system interact to change the states of the system over time. From these descriptions a modeler tries to identify the small subset of characteristics or features of the system that is sufficient to serve the specific objectives of the study.

(4) Conceptual Model Formulation - develop a preliminary model either graphically or in pseudo-code to define the components, descriptive variables, and interactions (logic) that constitute the system. This model formulation phase generates data input requirements for the model (Pritsker, 1986).

(5) Preliminary Experimental Design - select the measures of effectiveness to be used, the factors to be varied, and the level of those factors to be investigated.

(6) Input Data Preparation - identify and collect the input data needed by the model.

(7) Model Translation - formulate the model in an appropriate simulation language.

(8) Verification - establish that the computer program executes as intended.

(9) Validation - establish that a desired accuracy or correspondence exists between the simulation model and the real system. Carson (1986) and Sargent (1987) provides guidelines on the important issues of model verification and validation. Eklundh (1981) generalizes these techniques by discussing procedures for determining the accuracy of a simulation program.

(10) Final Experimental Design - design an experiment that will yield the desired information and determine how each of the test runs specified in the experimental design is to be executed.

(11) Experimentation - execute the simulation to generate the desired data and to perform the sensitivity analysis.

(12) Analysis and Interpretation - draw inferences from the data generated by the simulation.

(13) Implementation and Documentation - document the model and its use, record the findings, and implement the results.

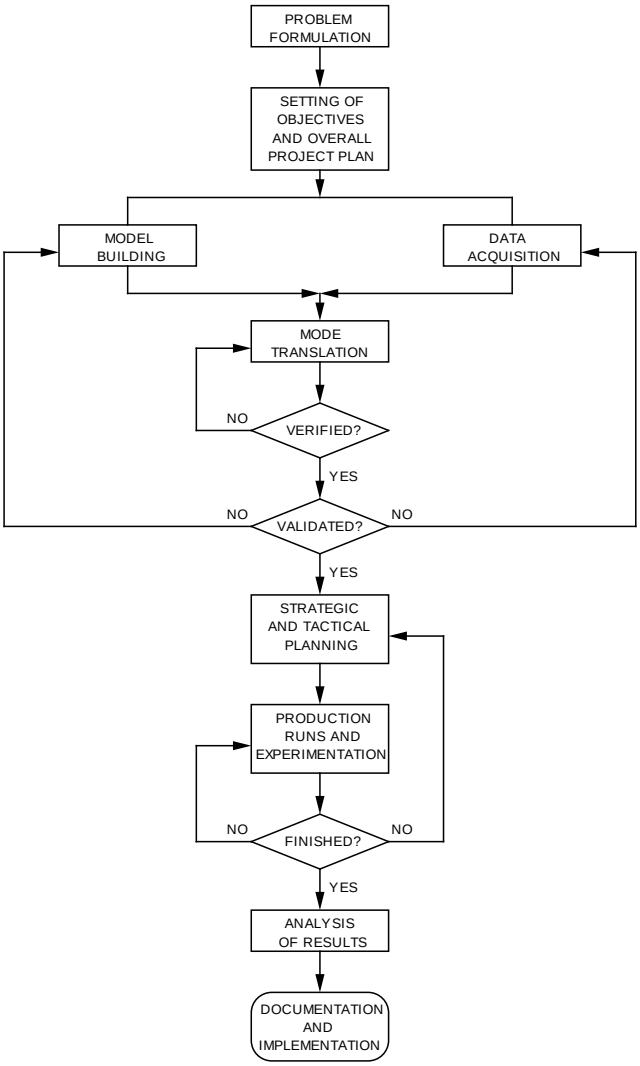


Figure 2.4. Steps in a simulation study [from Mackulak *et al.,* 1987].

2.1.4 Generating Random Variables

The underlying basis of the aggregation methodology that will be presented in Chapter 3 is its search for a procedure for generating random variables for an aggregation resource (a resource representing a combination of resources). To provide a foundation for this later discussion, it is important to first review the concept of random variables and discuss how simulation goes about generating their values.

A simulation model which has any random or stochastic aspects must sample or generate random variables from one or more distributions (Law and Kelton, 1982). While there are many techniques for doing this, the basic ingredient needed by them all is a source of independent, identically distributed uniform random variables with a lower bound of zero and upper bound of one.

Remarkably, random variates (or stochastic variates) from virtually any distribution can be obtained by transforming (0,1) random numbers. Generating truly random numbers is, in general, impractical and in fact undesirable (Arthur, 1989). Instead, generators of (0,1) pseudorandom numbers is needed. Fishman (1978) states that "much of the literature refers to pseudorandom number generation to emphasize that deterministic methods are used in practice. Generally, we omit the quantifier 'pseudo' both to save space and to avoid giving the reader an unnecessary feeling of discomfort for relying on such methods."

Arthur (1989) summarizes that the easiest methods of generating random numbers are performed by hand, such as by throwing a dice, dealing cards, or drawing balls from an urn. Next come random devices such as a rapidly spinning disk, pulsating vacuum tubes, and cosmic ray counters. Other techniques involve using a table of random numbers to using a decimal expansion for irrational numbers such as pi and e.

However, all of these approaches fail with regard to one or more of the following important considerations when selecting a random number generator (Arthur, 1989):

- The routine should be fast.
- The routine should not require a lot of computer storage.
- The routine should have a sufficiently long cycle before it repeats.
- The random numbers should be repeatable.
- The generated random numbers should closely approximate the ideal statistical properties of uniformity and independence.

In addressing the uniformity issue, Bratley, Fox, and Schrage (1983) use the following qualitative approach:

- Condition 1: The numbers when treated as points on the line segment for zero to one are approximately normally distributed.
- Condition 2: Successive nonoverlapping pairs when treated as point in the unit square are approximately normally distributed.

□

- Condition n: Successive nonoverlapping n-tuple of numbers when treated as points in the n-dimensional hypercube are approximately uniformly distributed.

Unfortunately, empirical test of this type are not statistically powerful and they are difficult to apply when n is greater than 10.

The technique most often used to generate random variates is to use a numerical scheme. A key characteristic is that these are almost always generated sequentially, with each new number being determined by one or more of its predecessors (Arthur 1989).

The *midsquare method* (Law and Kelton, 1982) is apparently the first proposed method for use on a digital computer. It starts with an initial number or seed, X_o. This number is squared, and the middle digits become U_1 (after placement of the decimal). The middle digits then also become X_1, and the process repeats. The major drawback of this approach is that it has poor statistical qualities and the seed needs to be chosen with care, for once zeroes appear, they are carried through in subsequent numbers.

The next general class of generators are the *additive congruential* and *Fibonacci generators* (Arthur, 1989). The general additive congruential generator adds two or more previous numbers together and then takes the remainder when this sum is divided by a number called the modules. The Fibonacci generator is a special case of this type in which the two most recent numbers are added together. The major disadvantage with this approach is that there is a lot of serial correlation.

The *linear congruential generator* is the most commonly used random number generator. Proposed by Lehmer (1951), it is defined by the recursive function:

$$X_i = (aX_{i-1} + c) \bmod m$$

The statistical quality of the generator is determined by the parameters a, c, and m and the seed X_o, such that:

$$0 \leq X_i < m \text{ and } U_i = X_i/m.$$

When c = 0, this results in a *multiplicative* generator and when c > 0, this is referred to as a *mixed* generator. This is the method of uniform random number generation in the SIMAN simulation language(Pedgen *et al.*, 1990).

As shown by Marsaglia (1968), the major drawback with this approach is that successive sequences of n numbers from a multiplicative generator all fall on at most $(n!m)^{1/n}$ parallel hyperplanes. For example, by plotting the sequences of three numbers as points in a cube and viewing the cube from the proper angle, then the points plotted appear as parallel lines.

Based on the work of Tausworthe (1965), a final type of generator, a *Tausworthe generator*, is an additive congruential generator of the form:

$$X_i = (a_iX_{i-1} + a_2X_{i-2} + \bullet\bullet\bullet + a_nX_{i-n} + c) \bmod m$$

where the modules m equals two. Because of this, each X_i can equal only zero or one. As such this, this type of generator produces a bit stream.

All of the above techniques are procedures for generating uniform zero-one random numbers. With a supply of these numbers, alternative algorithms can be used for generating random variables from a given distribution. Several factors should be considered when choosing the algorithm (Law and Kelton, 1991):

- *exactness* - The algorithm should generate random variables that follow the desired distribution better than the algorithms.

- *efficient* - The algorithm should minimize the required storage space and execution time.
- *complexity* - The algorithm should reflect the desired level of understanding. A more complex algorithm is appropriate if its operation need not be completely understood, otherwise a less complex (and more understandable) algorithm is more suitable.
- *robustness* - The algorithm should be efficient for all parameter values.

The first general approach to generating random variables having a continuous distribution is the *inverse transform method*. Ross (1985) remarks that this procedure is based on the following proposition:

> Let U be a Uniform (0,1) random variable. For any continuous distribution function F if we define the random variable X by
>
> $$X = F^{-1}(U)$$
>
> then the random variable X has distribution function F.

Law and Kelton (1982) summarize that to generate a random variable X from the continuous distribution F, when F^{-1} is computable, the following two steps are followed:

Step 1: Generate a random number U which is distributed as U(0,1)

Step 2: Set $X = F^{-1}(U)$ and return.

The second technique for generating random numbers is the *rejection method*. Ross (1985) explains that it assumes that a density function g(x) exists. With this, the basis for simulating from the continuous distribution f(x) is done by simulating Y from g and then accepting this simulated value with a probability proportional to f(Y)/g(Y). Specifically, let c be a constant such that

$$\frac{f(y)}{g(y)} \le c \text{ for all y}$$

With this, the technique simulates a random variable having density f.

The specific steps of the rejection method are (Ross, 1985):

Step 1: Simulate Y having density g and simulate a random number U from U(0,1).

Step 2: If $U \leq f(Y)/cg(Y)$ set $X = Y$. Otherwise return to Step 1.

The *composition* or *mixture technique* generates samples from distribution function F (or density function f) by using the fact that this function is a convex combination of other distributions $F_1, F_2,...F_n$ (or density function $f_1, f_2, ...,f_n$) (Law and Kelton, 1991).

Specifically, it is assumed that for all x, F(x) can be written as $F(x) = \sum_{j=1}^{\infty} p_j F_j(x)$ where $p_j \geq 0$, $\sum_{j=1}^{\infty} p_j = 1$, and each F_j is a distribution function. Correspondingly, the density function f can be written as $f(x) = \sum_{j=1}^{\infty} p_j f_j(x)$.

Sampling a random variable, X, using this method can be described in terms of three distinguishable steps . The first two steps are design steps and are performed only once. The third step is the procedure for sampling a value from the mixed or composite distribution. The first step requires the selection of (Peterson and Kronmal, 1982):

- the number $n \geq 1$ of elements in the mixture
- the mixture weights $p_1, p_2, ...,p_n$ ($p_i \geq 0$, i = 1,2,...,n; $\sum p_i = 1$), and
- the elements (density functions) $f_1, f_2,...,f_n$ of the mixture, subject to the constraint that the mixture of the density functions f_i, i = 1, 2, ..., n satisfy:

$$f(x) = \sum_{j=1}^{\infty} p_j f_j(x)$$

The second step of composite sampling involves selecting the methods for generating random variables. It requires the selection of: (Peterson and Kronmal, 1982)

- the method for generating the element identifier I, such that $P(I = i) = p_i$, and
- the methods for generating each of the X from f_i, i = 1,2,..., n.

This step identifies the random number generating technique to use for sampling a generating distribution and then the method for selecting a random variable from that selected distribution.

The final step is to develop the procedure for generating the random variables. For each sampling of X (a sample from the composite distribution F), the following two tasks are performed:

- Choose at random an element (distribution) of the mixture. That is, generate I such that $P(I = i) = p_i$.
- Generate a variable from the chosen resource distribution of the mixture. That is, generate a value for X from f_I.

This final step chooses the distribution function F_I with probability p_i. By conditioning on the value of I, the X (sample of F_I) returned by the algorithm will have distribution F (Law and Kelton, 1991). This procedure yields a random variable from distribution F since the random variable X has distribution

$$P(X \le x) = \sum_{i=1}^{\infty} P(X \le x | I = i) P(I = i) = \sum_{i=1}^{\infty} F_i(x) p_i = F(x).$$

Another general technique is the *hazard rate method* (Ross, 1985). There are also additional specific techniques for generating continuous random variables from certain continuous distributions (Normal, Cauchy, Gamma, Chi-Square, Exponential). The key to most of these techniques is through the use of *convolution*, where the desired random variable X is expressed as a sum of other random variables which are independently, identically distributed.

2.2 Discrete Manufacturing Systems

This research is concerned with developing an aggregate simulation model of a discrete part flow line manufacturing system. This section describes defines this type of system and explores the different types of performance measures that are sought by modeling it. Section 2.2.1 reviews the components and types of discrete manufacturing systems. Section 2.2.2 explores important performance characteristics of simulation models of systems manufacturing simulation studies.

2.2.1 Types of Discrete Manufacturing Systems

This section describes the system used for the processing and assembly of discrete products in large volumes. The operation involves the transformation of raw materials into goods that have value in the workplace (Groover, 1980). This transformation process involves a sequence of steps, with each step bringing the materials closer to a final state. These steps are called *production operations*.

Depending upon the nature of the production operation, there are two major types: manufacturing and process. Manufacturing industries are typified by discrete production operations (Groover, 1980), such as car or computer production. Process industries are typified by a continuous production processes, such as a chemical plant.

Within the area of manufacturing operations, there are several basic types (based on Groover, 1980):

(1) *Project.* This type of operation produces custom-made products. The total number delivered or produced is usually one. Examples include the production of a ship or space vehicle.

(2) *Job shop production.* The key characteristic of this type of production type is its low volume. Another distinguishing characteristic is that parts are processed in small batches. Examples include a machine tool shop or a commercial printer.

(3) *Mass production.* This operation involves the continuous manufacture of identical products. It is characterized by producing a high volume of similar parts. Examples include an automobile or toy manufacture.

(4) *Batch production.* This category involves the manufacture of medium-size lots of the same product. It is distinguished by producing a medium volume of periodic batches. Examples include furniture or textbook manufacturing operations.

As an interesting sidenote, Groover (1980) remarks that "as much as 75% of all part manufacturing is in lots of size 50 pieces or less." He continues and concludes that batch production and job shop production constitute a majority of manufacturing activity.

A special case of mass and batch production are manufacturing flow lines. Aneke and Carrie (1984) develop a classification for flow line systems based on the number of product types, shop layout and flow characteristics. They concluded that there exists ten feasible systems (based on Aneke and Carrie, 1984):

(1) *Single product single machine system* - a single product is completely produced by a single machine.

(2) *Mixed product single machine system* - the same type of operation is required by a number of different parts.

(3) *Multi-product single machine system* - different products are produced from time to time on a single machine.

(4) *Single product sequential flow line* - a single product requires more than one machine. This leads to machines being arranged in a line such that the operational sequence of all parts is the same.

(5) *Mixed product sequential flow line* - all the parts have the same operational sequence with no need for tool resetting.

(6) *Multi-product sequential flow line* - all the parts have the same operational sequence but are produced separately in batches.

(7) *Mixed product bypass flow line* - the operational sequence of parts may vary by omitting operational steps.

(8) *Multi-product bypass flow line* - a batch may vary from the operational steps only by omitting an operation.

(9) *Multi-product backtracking flow line* - a batch can vary from the operational steps by either omitting a step and/or backtracking to a previous operation. This product flow line is bi-directional.

(10) *Multi-product multi-directional backtracking system* - even though the products are batched, the operational sequences are so varied that flow is multi-directional. An example of this operation is a job shop.

As a means to describe discrete manufacturing system, Dietrich (1991) proposes a classification scheme based on the production process, the system management, and the system behavior. The production process is the planned sequence of operations required to produce a finished part and its specification defines the physical components of a manufacturing system. These physical components are divided into the following four categories:

(1) *production process* - the planned sequence of operation required to produce a part. It consists of the specific operations, the operation sequence for the parts, and the distribution of operations among the machines, tools, and operators.

(2) *material flow* - the flow of material in a manufacturing systems. It is composed of the possible routes for a part and each routes' consistency (probability of being operational).

(3) *information system* - the information required by the manufacturing system is stored in and communicated by a single system. This system consists of a single integrated data base or several disjoint ones.

(4) *contention for resources* - unless the availability of all the system resources far exceeds the amount required to produce the finished part, there will be competition for the resources. Examples include contention for machines, contention for material and contention for material handling.

Similar to Dietrich's work, Ozdemirel (1990, 1993) develops a classification scheme for discrete manufacturing. Her research divides the physical system characteristics into the following components:

(1) *Work stations* - any system resource that is not a material handling device. Of the four types, machining stations process a part, assembly stations perform assembly operation on a part, inspection stations inspect a part, and load/unload stations are where parts arrive and leave the system.

(2) *Material handling* - provide a mechanism which allows parts to move from one work station to another. Transports can be done manually, with an automatic guided vehicle, or a conveyor system.

(3) *Jobs* - characteristics of the part(s) being processed in the manufacturing system.

(4) *Work-in-process storage areas* - buffer areas where parts wait for service on a machine.

(5) *Equipment breakdown and scheduled maintenance* - the specification of breakdown and maintenance that the system experiences.

2.2.2 Performance Measures for Simulation Studies

With simulation, a system wide view of the effects or changes in the manufacturing system can be explored. Specific (potential) benefits of simulation in manufacturing are increased throughput, reduced in-process inventories, increased utilization of machines and workers, increased on-time deliveries and reduced capital requirements.

Ozdemirel (1990, 1993) develops a classification scheme for discrete part manufacturing systems. To extend her classification scheme to include simulation, her research identifies a set of user goals of objectives for performing a simulation study of a discrete part manufacturing system. There are three major goals for a simulation study (Ozdemirel 1993):

- prediction
- scheduling alternatives
- optimization

For predictive simulation studies, there are six specific types of items to study: job volume, effect of hot jobs, bottleneck resources, breakdown effects, product quality, and absenteeism effect. Models exploring scheduling alternatives look at the impact of product mix, sequencing alternatives and push versus pull inventory systems. Models which seek to optimize a manufacturing system to optimize the fixed shop floor structure or optimize structural changes.

With fixed shop optimization, specific ideas can be explored: lot size verses setup time, utilization verses cycle time, minimization of buffer stocks, minimization of work-in-process, and input data accuracy and sensitivity for fine tuning of the model. A model seeking to optimize structural changes explores: optimizing work station layout, optimizing use of material handling equipment, optimizing physical work-in-process areas, optimizing the use of secondary resources and the general layout of the facility.

To measure the effectiveness in meeting the above modeling objectives, the system must be measured. Common measures of performance used in manufacturing simulation studies must be included (Law 1986):

- Throughput (number of parts produced per unit of time).
- Time parts spend in queues.
- Time parts spend being transported.

- Time parts spend in the manufacturing system.
- Sizes of in-process inventories.
- Utilization of equipment and personnel.
- Proportion of time that a machine is broken, blocked, or starved.
- Proportion of parts which must be reworked or scrapped.
- Return on investment for a new or modified manufacturing system.

Although not exhaustive, this list indicates the most common measures of performances. The objective of this research is to estimate the cycle time (sojourn time) of a part. This is the total time a part spend in the manufacturing flow line waiting and being serviced. As outlined in Chapter 1, this quantity is important for estimating completion dates and can be used for modeling the financial cost of parts currently in progress.

2.3 Fundamentals of Queueing Theory

The terms “waiting line” or “queue” (or resource) are used to characterize a broad class of stochastic processes with three primary characteristics which can describe a system (Giffin, 1978):

- *An input process*. A process which can be described by a random variable which explains how a part or customer arrives to a queue. Wagner (1969) explains that usually the pattern of arrivals into a queue system is described by a probability distribution of time between successive arrival events, and the number of individuals or parts that appear at each of these events. Defining the arrival distribution involves determining how many parts arrive and the pattern of arrivals over a given period of time (Anderson, *et al.*, 1988).
- *A service mechanism*. A process which can be described by a random variable which explains how parts or customers are serviced once leaving the queue. A specification of the service mechanism includes a description of the time to complete a service, and of the number of individuals or parts whose requirements are satisfied at each service event (Wagner, 1969).
- *A queue discipline*. The discipline defines the rules of behavior within the queue, and how parts or customers are selected for service from the queue. Anderson, *et al.* (1988) summarizes this as “the manner in which waiting parts are ordered for service”.

Though the following two components are implied in the above listing, Lee (1988) specifically separates them:

- *A waiting line*. When parts arrive in such a way that they have to wait for service, waiting lines, or queues will develop.
- *Departures*. Once arrivals are serviced, they become departures and leave the queueing system.

These characteristics describe a system whose performance can be modeled with queueing theory. Queuing theory studies systems by formulating mathematical models of their operation and then using these models to derive measures of performance (Hillier and Leiberman, 1986).

Queueing problems have been actively researched since 1907 (Giffin, 1978). The first published work occurred when A.K. Erlang published his fundamental papers on congestion in telephone traffic (Erlang, 1909). This large history results in queueing theory being one of the oldest studied problems in operations research (Lee, 1988).

In 1953, D.G. Kendall introduced a compact notation scheme to describe the characteristics of a queueing system (Lee, 1988). The Kendall notation for describing a queue is (Giffin, 1978):

arrival process / service process / number of parallel servers

When necessary an appendage may be added of the form

/ limit on the number in the system / number in the source / queue discipline

If the appendage is omitted it is assumed that there is no limit on the system capacity, customers are drawn from an infinite population, and the queue discipline is first come, first serve.

The uncertainty present in most real systems is what makes model building a challenging task. The solution is to describe the arrival and service processes by their interarrival and service time probability distributions (Giffin, 1978). The number of servers is an integer. Standard queue disciplines are identified by appropriate abbreviations. A common notation is (Giffin, 1978; Mehdi, 1991):

M Exponential

E_k Erlangian (gamma) with k identical phases

D Constant (Deterministic)

G General

GI General independent distribution of interarrival time (also referred to as G)

H Hyperexponetial

PH Phase type

S Number of servers in parallel

FCFS First come, first served

LCFS Last come, first served

RSS Random selection for service

PR Priority

Molloy (1989) provides the following additional queue selection rule:

RR Round robin, where a small, fixed amount of service is given to each part in a circular fashion.

To illustrate the notation,

M/M/1 Markovian input and service with one server,

M/G/1 Markovian input, general service distribution, and one server,

G/G/S General input, general service distribution, and S servers,

are all descriptions of queueing systems.

Plane and Kochenberger (1972) remark that a key term in the analysis of a queuing system is the *state* of the system. They define this as the number of customers in the entire queuing facility system which includes customers both waiting in line and in the service facilities. Once the probability of finding the system in any given state is known, it is easy to derive characteristics of the system. Plane and Kochenberger (1972) present the following list of commonly explored system operating characteristics:

- the expected number of customers in line
- the expected number of customers in the system
- the expected waiting time in the line
- the expected service time

• the expected time in the system

There exists a simple relationship between the expected number of parts present under steady state conditions and their expected waiting times:

$$L = \lambda W.$$

This relationship is referred to as Little's formula (Giffin, 1978). It states that the average number of customers in the system is equal to the average arrival rate multiplied by the average amount of time spent by the customers in the system (Little, 1961). The significance of this is that a specific arrival or departure process need not be specified. The only assumption is that no work is created or destroyed within the queue system (Molloy, 1989). Therefore, Little's formula is valid for a wide range of systems.

An important aspect of queuing theory is that the results give the steady state behavior of the system. The queueing system initially starts in a transient state. During the course of time it approach equilibrium or steady state conditions. That is, over time the system approaches a steady state, in which the characteristics of the system are no longer dependent upon the length of time since start-up. If the part arrival rate is greater than or equal to the service rate times the number of servers (this quantity is defined as ρ), steady state conditions cannot exist (Plane and Kochenberger, 1972). Theoretically, when $\rho \geq 1$, the queue length grows without bounds. Mehdi (1991) proves the general relationship that in any queueing system in which arrivals occur one by one and that has reached steady state, the probability that an arriving part finds *n* parts in the system when it arrives is equal to the probability that a departing part find *n* in the system. The development of all the queueing results of this research is based on the assumption that the queues (or resources) of the flow line are in steady state.

The remainder of this section explores the development of mathematical models for estimating the performance of specific queueing systems. Section 2.2.1 considers the case of M/M/S systems. Section 2.2.2 reviews M/G/S systems. Section 2.2.3 explores

approximations for the G/G/S systems. Sections 2.2.4 addresses special conditions that can occur in a queueing system. Section 2.2.5 explores the procedures and evaluation of networks of queues. Finally, Section 2.3.6 explores a special queueing network called tandem queueing systems.

2.3.1 Exponential Arrivals and Exponential Service

There is a class of queueing models for which detailed solutions are quite easy to obtain (Ravindran, *et al.,* 1987). In this class of systems, both the interarrival times and service times are negatively exponentially distributed. The advantage of this distribution family is that it can be characterized by a single parameter, namely its mean.

The first queue of this system to explore is the M/M/1. For a queueing system to be defined of this type it must meet several assumptions (Anderson *et al.,* 1988):

- The queue has a single server (or channel)
- The pattern of arrivals to the queue follows a Poisson probability distribution
- The service times follows an exponential probability distribution
- The queue discipline is first come, first serve (FCFS)

The model describing the performance of a M/M/1 queueing systems is (Lee, 1988):

λ Mean arrival rate

$1/\lambda$ Mean time between arrivals

μ Mean service time

$1/\mu$ Mean service time

ρ Traffic intensity: $\rho = \frac{\lambda}{\mu}$

P_o Probability the system is empty: $P_o = 1 - \rho$

P_n Probability of n parts in the system: $P_n = \rho^n(1-\rho)$

L_q Expected number in the queue, excluding the parts being serviced, under steady state conditions. Mean queue length: $L_q = \frac{\lambda\rho}{\mu - \lambda} = L\rho$

L Expected number in the system, including the parts being serviced, under steady state conditions. Mean system length: $L = \frac{\lambda}{\mu - \lambda}$

W_q Expected time spent in the queue, excluding service time, by a part at steady state. Mean waiting time: $W_q = \frac{\rho}{\mu - \lambda}$

W Expected time spent in the system, including service time, by a part at steady state. This is also known as the sojourn time or the cycle time. Mean time in the system: $W = \frac{1}{\mu - \lambda}$

A logical extension to the single-server case is a queuing system with multiple servers. The assumptions required of the M/M/S queueing system are: (based on Giffin, 1978)

- Parts arrive in a Poisson fashion at rate λ.
- There are S identical servers in parallel.
- The service time distribution of each server is exponential with a mean of $1/\mu$.
- Servers are noncooperative. That is, one part is never attended by more than one servers.
- Arriving customers are serviced by the first available server. When all servers are busy, parts form a single queue with unlimited waiting space from which they are served in a FCFS fashion.

The M/M/S can be described by the following model (Lee, 1988):

λ Mean arrival rate

$1/\lambda$ Mean time between arrivals

μ Mean service time

$1/\mu$ Mean service time

S Number of parallel, identical servers

ρ Traffic intensity: $\rho = \frac{\lambda}{S\mu}$.

P_o Probability the system is empty: $P_0 = \frac{1}{\left\lceil \sum_{n=0}^{S-1} \frac{(\lambda/\mu)^n}{n!} \right\rceil + \left\lceil \frac{(\lambda/\mu)^S}{S!(1-\rho)} \right\rceil}$

P_n Probability of n parts in the system: $\begin{cases} P_n = \frac{(\lambda/\mu)^n}{n!} & n \le S \\ P_n = \frac{(\lambda/\mu)^n}{S!S^{(n-S)}} & n > S \end{cases}$

L_q Mean queue length: $L_q = \frac{\lambda\rho}{\mu - \lambda} = L\rho$

L Mean system length: $L = L_q + \frac{\lambda}{\mu}$

W_q Mean waiting time: $W_q = \frac{L_q}{\lambda}$

W Mean time in system: $W = W_q + \frac{1}{\mu}$

The results for the M/M/1 queueing system are merely a reduction and simplification of the M/M/S formula for the special case when the number of servers is equal to one. An interesting aspect of a M/M/S queueing system in steady state with arrival rate λ and service rate μ is that the interdeparture times (time between parts leaving the queue) are independently and identically distributed as an exponential random variable with mean $1/\lambda$. In other words, the output process of a M/MS queue system is Poisson with the same parameter as the input process. This was formally proven by Burke (1956). In addition, Rao and Posner (1984) expand this work by exploring the output process of an M/M/1 queue with randomly varying system parameters.

A major criticisms of queuing theory includes the absence of statistical procedures and the indiscriminate use of M/M/S models and steady state results. Budnick, Mojena, and Vollman (1977) claim that all too often, a disconcerting absence of proper sampling,

estimation, and hypothesis testing procedures are performed. All assumptions inherent in a model (for example, types of distributions, parameters, state-independence, independent arrivals, and so forth) must be tested in order to select the correct model. While M/M/S models are widely available and useful, their assumptions must be reasonably substantiated for a particular application. The limitations and corresponding interpretations of steady state solutions must be assessed, as many queuing systems do not operate a sufficiently long period of time to achieve steady state. The decision to base conclusions on steady state solutions is not necessarily unjustified, but rather their decision should represent the most idealized case.

2.3.2 Exponential Arrivals and General Service

The next system to study is one with exponential arrivals and general service times. Studying these systems is aided by the fact that one of the distributions is negatively exponentially distributed. As such, the solution procedure uses a concept of "embedding a Markov chain" (Ravindran *et al.,* 1987). This procedure involves viewing the process only at selected moments and ignoring the dynamic behavior of the process at intermittent times. The moments at which the process is viewed, the embedded points, are chosen so that the discrete-time Markov assumption will hold. That is, given the state at one of these points, enough information is available to predict the state at the next point.

The easiest system to study is the M/G/1 queue. In such a system, a Markov chain is embedded at the moments of service completion. At these times, and only at these times, it is necessary to know the number in the system to predict the future. Giffin (1978) discusses the development of the M/G/1 and shows that the resulting performance measures are:

λ Mean arrival rate

$\bar{t}$ Mean service time

σ_t^2 Variance of the service time

L Mean system length: $L = \lambda\bar{t} + \frac{(\lambda\bar{t})^2 + \lambda^2\sigma_t^2}{2(1-\lambda\bar{t})}$

L_q Mean queue length: $L_q = \frac{\lambda^2(\bar{t}^2 + \sigma_t^2)}{2(1-\lambda\bar{t})}$

W_q Mean waiting time: $W_q = \frac{\lambda(\bar{t}^2 + \sigma_t^2)}{2(1-\lambda\bar{t})}$

W Mean time in system: $W = \bar{t} + \frac{\lambda(\bar{t}^2 + \sigma_t^2)}{2(1-\lambda\bar{t})}$

The important measures of performance can be evaluated with very limited knowledge, namely the first two moments of the service distributions. The formula for L is referred to as the Pollacezek-Khintchine (P-K) formula (Gross and Harris, 1974; Giffin, 1978). It permits estimates of the expected line length in M/G/1 systems from the knowledge of the arrival rate and the mean and variance of the service distribution.

A benefit of this formula is that when the service time is exponential, $\bar{t} = 1/\mu$ and $\sigma_t^2 = 1/\mu^2$, the formula reduces to the formula for the M/M/1 model (Hillier and Lieberman, 1986).

An exact formula for the case of two or more servers is not known. But several authors (Hokstad, 1978; Stoyan, 1976) have proposed an approximation of the M/G/S queueing system:

λ Mean arrival rate

μ Mean service rate

$\bar{t}$ Mean service time

cv_t^2 Squared coefficient of variation of the service time

S Number of parallel, identical servers

ρ Traffic intensity: $\rho = \frac{\lambda}{S\mu}$

C The probability that an arriving part has to wait for service: $C = \frac{(\lambda \bar{t})^S}{S(1-\rho)} P_0$. This is often referred to as Erlang's C Formula (Mehdi, 1991).

P_0 The probability that zero service are busy: $\frac{1}{\left[\sum_{n=0}^{S-1} \frac{(\lambda \bar{t})^n}{n!} + \frac{(\lambda \bar{t})^S}{S!(1-\lambda \bar{t}/S)}\right]}$.

$E_{M/G/1}[W_q]$ Expected queue waiting time for M/G/1 queue:

$$E_{M/G/S}[W_q] = \frac{1+cv_t^2}{2\lambda(1-\rho)} \rho C$$

The key advantage of this approximation is that it is exact for M/M/S and M/G/1 queueing systems. Several other approximations of the M/G/S system have been proposed (Takahashi, 1977; Boxma *et al.*, 1979; Tijms *et al.*, 1981; and Kimura, 1986).

2.3.3 General Arrivals and General Service

The "embedding a Markov chain" idea does not work for the G/G/S queue because "the only possible embedding points are those moments in which an arrival and a service completion coincide exactly, but these moments are so rare that they may be considered useless for modeling purposes" (Ravindran, *et al.*, 1987). As such, only approximations are available for estimating the performance characteristics of the G/G/S queueing system.

By interpolating the G/M/S and M/D/S queues, Kimura (1991) obtains a distribution-dependent approximation for the mean waiting time in the G/G/S queue:

λ Mean arrival rate

cv_a^2 Coefficient of variation of the arrival time

μ Mean service rate

cv_t^2 Coefficient of variation of the service time

S Number of parallel, identical servers

ρ Traffic intensity: $\rho = \dfrac{\lambda}{S\mu}$

g $\quad g\left(\rho, cv_a^2, cv_t^2\right) = \begin{cases} Exp\left[-\dfrac{2(1-\rho)}{3\rho} \dfrac{(1-cv_a^2)^2}{cv_a^2 + cv_t^2} \right], & cv_a^2 \le 1 \\ 1, & cv_a^2 > 1 \end{cases}$

g_1 $\quad g\left(\rho, cv_a^2, 1\right)$

υ $\quad \upsilon\left(s, cv_a^2, cv_t^2\right) \approx 1 + \dfrac{S}{S-1}\,\eta\left\{ \dfrac{cv_a^2 + cv_t^2}{S\mu I(S)} - (cv_a^2 + 1) \right\}$

I(S) $\quad I(s) = \left\{ \left(s + \dfrac{1-cv_s^2}{1+cv_s^2} \right)\mu \right\}^{-1}$

η $\quad \eta\left(cv_a^2, cv_s^2\right) = \begin{cases} 1.1Exp\left[\dfrac{-2.4(1-cv_a^2)}{cv_a^2} \right] - 0.1 & cv_a^2 \le 1 \\ Min\left(.35(cv_a^2 - 6), 1\right) & cv_a^2 > 1 \end{cases}$

$E_{G/M/S}[Wq]$ Expected queue waiting time for a G/M/S queue

$E_{M/D/S}[Wq]$ Expected queue waiting time for a M/D/S queue

$E_{G/G/S}[Wq]$ Expected queue waiting time for a G/G/S queue:

$$E_{G/G/S}\left[W_q\right] \approx (cv_a^2 + cv_s^2) g \left\{ \frac{(cv_a^2 + 1) g_1 \upsilon}{EWq(G/M/S)} + \frac{1-\upsilon}{EWq(M/D/S)} \right\}^{-1}$$

This formula is for the case of the squared coefficient of service time variation being less than or equal to one. Kimura proposes a similar formula for when it is larger than one. To examine the performance of this estimate, Kimura (1991) carried out numerical experiments and compared them with the exact results and previous two-moment approximations. The results indicate that the relative percentage errors are on the order of 5% in moderate traffic and 1% in heavy traffic.

As a special case of the G/G/S results, Kumura (1991) shows that for the G/G/1 system, the approximation reduces to:

λ Mean arrival rate

cv_a^2 Coefficient of variation of the arrival time

μ Mean service rate

cv_t^2 Coefficient of variation of the service time

ρ Traffic intensity: $\rho = \frac{\lambda}{\mu}$

g $$g(\rho, cv_a^2, cv_t^2) = \begin{cases} Exp\left[-\frac{2(1-\rho)}{3\rho}\frac{(1-cv_a^2)^2}{cv_a^2 + cv_t^2}\right], & cv_a^2 \le 1 \\ 1, & cv_a^2 > 1 \end{cases}$$

$E_{M/M/1}[Wq]$ Expected queue waiting time for a M/M/1 queue:

$$E_{M/M/1}[W_q] = \frac{\rho}{\mu - \lambda}$$

$E_{G/G/1}[Wq]$ Expected queue waiting time for a G/G/1 queue:

$$E_{G/G/1}[W_q] \approx \frac{cv_a^2 + cv_t^2}{2} g\{E_{M/M/1}[W_q]\}$$

Kimura (1986) remarks that this approximation matches that proposed by several other authors (Whitt, 1983; Kramer and Langenbach-Belz, 1976).

Due to G/G/S queues have not having defined solutions, much research effort has gone into estimating performance bounds. For instance, Daley and Rolski (1992) studied light traffic approximations in many server queues, Seelen and Tijms (1984) approximated the conditional waiting times in the G/G/S queue, and Suzuki and Yoshida (1970) explored the specific case of a G/G/2. Marchal (1978) proposes and Kleinrock (1976) presents the following bounds of a G/G/S queue:

λ Mean arrival rate

cv_t^2 Squared coefficient of variation of the service time

σ_a^2 Variance of the interarrival time

μ Mean service rate

$\bar{t}$ Mean service time

$\overline{t^2}$ Second moment of the service time

σ_t^2 Variance of the service time

S Number of parallel, identical servers

ρ Traffic intensity: $\rho = \frac{\lambda}{S\mu}$

$E_{G/G/S}[W_q]$ Expected queue waiting time for G/G/S queue

Thus, the range of the expected waiting time is:

$$\frac{\rho^2 cv_{\bar{t}}^2 - \rho(2-\rho)}{2\lambda(1-\rho)} - \frac{\left[\frac{(S-1)}{S}\right]\bar{t}^2}{2\bar{t}} \le E_{G/G/S}[W_q] \le \frac{\sigma_a^2 + (1/S)\sigma_{\bar{t}}^2 + \left[\frac{(S-1)}{S}\right]\bar{t}^2}{2\bar{t}(1-\rho)}$$

As with the general server case, much research has looked on bounding the expected waiting time (Shanthikumar, 1983; Mori, 1975) of a single server system. Kingman (1962) showed that the behavior of the G/G/1 queue in a heavy traffic case (*i.e.*, $\rho \cong 1$), the waiting time distribution can be approximated by an exponential distribution such that the mean waiting time is: $W_q \cong \frac{\left(\sigma_a^2 + \sigma_{\bar{t}}^2\right)}{2\bar{t}(1-\rho)}$. Marshall (1968) recounts that for the G/G/1 system, the bounds on the expected waiting time for a G/G/1 queue are:

λ Mean arrival rate

σ_a^2 Variance of the interarrival time

μ Mean service rate

$\bar{t}$ Mean service time

$\sigma_{\bar{t}}^2$ Variance of the service time

ρ Traffic intensity: $\rho = \frac{\lambda}{\mu}$

$E_{G/G/1}[W_q]$ Expected queue waiting time for G/G/1 queue

The resulting range on the expected waiting time is:

$$\frac{\lambda^2 \sigma_{\bar{t}}^2 + \rho(\rho-2)}{2\lambda(1-\rho)} \le E_{G/G/1}[W_q] \le \frac{\lambda\left(\sigma_a^2 + \sigma_{\bar{t}}^2\right)}{2(1-\rho)}$$

2.3.4 Queueing Variations

There are many conditions which can be added to embellish the base queueing system model. Possible changes that can occur include: changes to the arrival process,

placing restrictions on the service capacity, or having a different service operation. Figure 2.5 (from Saaty, 1961) is an abbreviation of the many variations and special conditions that can occur in queueing systems. It attempts to show the various possibilities that influence arrivals, the times of arrivals, the queue and different types of queue discipline, the service channels, and the output.

Saaty p12

Figure 2.5. The various conditions and special cases which can occur in queueing systems [from Saaty, 1961].

Service and Arrival Distributions

Up to this point, this section has discussed only a limited number of queueing models (M/M/S, M/G/S, and G/G/S). There are many other distributions which can be studied. For example, a G/M/S system is one with general arrivals and exponential

service. For the special case of the G/M/1 queue, the concepts of "embedding a Markov chain" can be used, where the embedding occurs at the moments in which arrivals occur (Ravindran *et al.,* 1987). Unfortunately, Plane and Kochenberger (1972) remark that even though the assumption of exponential service times is a reasonable assumption for real world systems, it has a significant disadvantage in that the exponential distribution assumes that the most likely or modal service time is zero. In other words, zero (or very small service time) is the most likely to occur. This is unlikely to be true in most systems.

Finite Storage Capacity

Although it is mathematically convenient to assume that a queueing system has infinite storage capacity, as a practical matter very few physical systems can meet this specification (Giffin, 1978). In actuality, because the arrival process will continue to generate arrivals even when the system is full, those parts that arrive during this condition are "blocked" arrivals and are lost to the system (Ravindran *et al.,* 1987). Page (1972) recounts that most practical problems have a limit on the number of parts that may queue at any given time. If this limit is larger than any likely size of the queue it is valid to assume the queue is an infinite queue. Otherwise, a separate study with the limit condition applied must be performed. The mathematics of modeling this condition when a system reaches its capacity or truncation point involves reducing the arrival rate to zero until such time as a customer is served to again make queue space available.

Finite Population

The next queueing extension concerns the case in which there is a limited number of customers in the calling population. Such a situation exists when the potential arrivals form a fixed, finite population (Ravindran *et al.,* 1987). Carmichael (1987) discusses that

the calling population can either be infinite or finite in size. For finite, but large, calling populations "the assumption of being infinite in size is often made as this case is mathematically the more tractable and predominates the literature."

Bulk Queues

This type of queue is concerned with parts which arrive in a group. In a manufacturing system, rather than have parts arrive singularly to a queue, they may arrive grouped as a batch.

Queue Discipline

In every model so far discussed, the implied priority scheme has been first come, first serve (FCFS). With such a queue discipline, the part that has been waiting the longest is the next to be selected for service. Giffin (1978) states that other commonly encounters queue disciplines are last come, first serve (FCFS) and random selection for service (RSS). A benefit of the wide applicability of Little's formula is that is valid for the LCFS and RSS cases (Ravindran *et al.*, 1987). Gnedenko and Kolvalenko (1989) show that as a corollary to Little's Theorem of $L = \lambda W$, in a GI/G/S system under steady state conditions, the mean waiting time (duration time in the system) does not depend on the queue selection rule.

A queue can also allow certain part types a priority over other types in the system. In such a system, certain parts types have priority of the service mechanism and can interrupt work on a less priority part type.

State-Dependent Rates

The final illustration concerns altering the service and arrival rates as functions of the number of customers in the system (Giffin, 1978). Page (1972) remarks that

"customers or parts may be discouraged from joining the queue when it is large because of the time required for waiting to get served. In fact, the chance of a customer not joining a queue should intuitively increase as the queue gets larger. That is, the instance of customers "balking" increases with the queue size.

2.3.5 Networks of Queues

To this point only queuing systems that have a single service facility with one or more servers have been considered. That is, every customer or part demands one service and leaves the system as soon as it is obtained (Medhi, 1991). In many situations, the queues will not occur in isolation but as part of an organized system. Ravindran, Phillips, and Solberg (1987) give the following example: a factory ordinarily contain dozens of queues, linked together by the logical sequence of the production process. Systems such as this often contain complicated behavior resulting from:

- the direct interaction of the arrival and service processes,
- branching,
- merging,
- and looping of traffic streams.

Hillier and Leiberman (1986) recount that many queuing systems encountered in operations research studies are actually queuing networks. In such a network of service facilities, customers must receive service at some or all of these facilities. They provide the following example:

> Orders being processed through a job shop must be routed through a sequence of machine groups (service facilities). It is therefore necessary to study the entire network to obtain such information as expected total waiting time, expected number of customers in the entire system, and so forth.

Perhaps the most obvious approach to dealing with networks of queues is to separate them into subsystems, each which has only one queue, and then analyze each subsystem individually (Ravindran, Phillips, and Solberg, 1987). Such an approach permits the use of the wide variety of available models for single queues, while immediately extending the range of the applications to systems of arbitrary size. Ravindran *et al.* (1987) states that there are two problems with this approach: (1) it does not always work in obtaining valid results, and (2) even when it is technically correct, it neglects the interactions among the queues, which is often the most critical aspect affecting network behavior.

There is a class of queueing networks for which the decomposition strategy works well. Provided that all of the required conditions are satisfied, each queue and its associated servers will act, and can be modeled, as if they were an independent Markovian queue of the M/M/S type. The required conditions are:

- All external arrivals to the network occur in independent Poisson streams. There may be several different streams entering at different points.
- All service times are negatively exponentially distributed with rates that depend at most upon the local queue.
- All queues have unlimited capacity. Blocking and overflow are not permitted.
- Any branching of the internal traffic stream is probabilistic, with probabilities that are independent of everything except the position in the network. In other words, after completing service at subsystem i, a customer would go next to subsystem j with probability p_{ij}.

Any network of queues satisfying these conditions is said to be an open network having a product form solution. The *open* part of the description refers to the fact that arrivals from the outside are accepted, and the *product form solution* refers to the idea that the network factors into independent subsystems. The analysis of each subsystems

amounts to straightforward application of the M/M/c results of queuing theory. The only step that may not be obvious is the calculation of the net arrival rate to each subsystem, since it comprises (possibly) a direct external arrival stream and several internal traffic streams coming from other subsystems (Ravindran *et al.*, 1987).

Before algorithms used to solve queuing network systems are presented, the problem must first be characterized. In a queueing network, the input process is often a merging of the output processes of other queues. To begin, it is further necessary to characterize the output process of a single queue.

The output of a queue is just another stochastic process (Molloy, 1989). The output process depends on the input process, the queueing discipline, and the service process. When studying the output of a queue, a particular input process must be determined. It can reasonably be assumed that the input process to a queue is Poisson. If the output process is again Poisson, we say that the queue has the M □ M (Markov to Markov) property (Mehdi, 1991).

Since a queue delays arrivals, it will clearly change the timing and possibly the order of the arrivals (Molloy, 1989). The output process is not the same process as the input process. However, if the input process to a queue having the M □ M property is Poisson, the output process will be another Poisson process with the same parameter (Burke, 1956). That is, the interdeparture time for an M/MS system is the same as the interarrival time distribution. This is intuitive in that the mean output or departure rate should be the same as the mean input rate during steady state. Hence, a M/M/S queueing system has no impact on the arrival process of other queues in the network.

Carmichael (1987) explains that there is an alternative approach to viewing this independence. For example, in a two stage system, $P(n_1,n_2) = Pn_1Pn_2$. That is, the probability of there being n_1 customers in stage 1 and n_2 customers in stage 2 is the product of the individual probabilities.

There are queues (for instance, the M/M/S) that preserve the Poison nature of their inputs. For instance, if the input of a M/M/1 queue is the output of another M/M/1 whose inputs are Poisson, then it is possible to solve the system (Molloy, 1989). Since a M/M/1 queue has the M□M property, a network of such queues can be solved by solving for the distribution of each queue independently as long as the network is feedforward. Unfortunately, if feedback occurs, one of the assumptions is violated.

With feedback, it can no longer be assumed that the input to the queue is Poisson. Although it is true that merging independent Poisson processes results in a Poisson process. Molloy (1989) presents the following the following illustration (Figure 2.6) of the simplest feedback system.

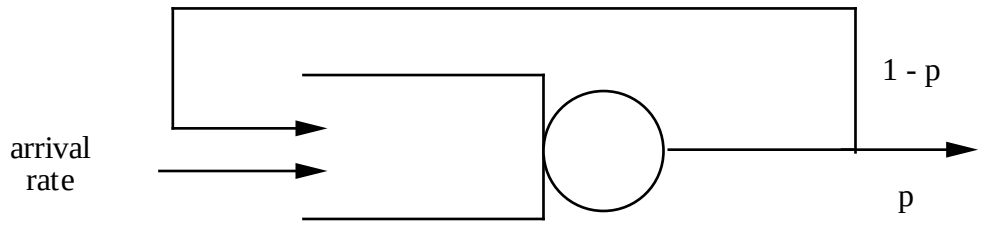


Figure 2.6. A queue with feedback [from Molloy, 1989].

To understand that the two processes feeding the queue (the Poisson process from the outside world and the feedback process) result in a non-Poisson process, consider the residual life of an arrival. Molloy (1989) explains that if the input process were Poisson, it would be memoryless. But the input process depends on the state of the queue because of the feedback. The state of the queue has memory of the most recent arrivals, so the feedback process has memory of the previous arrivals. Therefore, it cannot be Poisson.

To further understand this, consider the following example from Molloy (1989): assume that the probability of returning to the queue is close to one and therefore the service rate is much larger than the arrival rate. As the system runs, the input process is

dominated by the feedback (looking much like the service process) until the queue empties, at which point the input process is simply the slow, outside arrival process.

With this understanding of the key issues necessary for a queuing network, consider networks which have the following characteristics (Giffin, 1978):

- The networks contain more than one service center
- Each service center is a multi-channel queue with each channel at that center having an identical exponential service time distribution.
- Arrivals at any given center may come from outside the system or from other centers in the network.
- Arrivals from outside the network occur in Poisson fashion.
- When a unit completes service at a particular center it may leave the system or be routed to another center, its path being controlled by a fixed probability distribution associated with the center it is leaving.
- There is unlimited waiting space at every service center.
- Total arrival rate at every center is less than its potential service rate.

Networks satisfying this set of characteristics are called *Jackson networks*. Note that this listing is nearly identical to the previously list of characteristics.

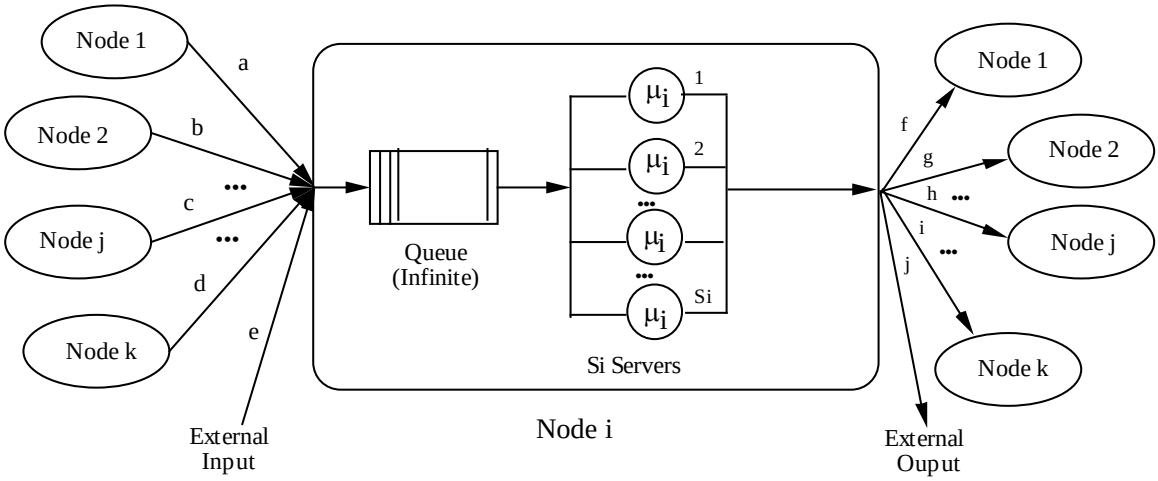
A more formal definition of a Jackson network is given by Mehdi (1991). Parts from (say) node i proceed to an arbitrary node and new customers may join node i from the outside. Suppose there are k nodes, where the ith node (i = 1 to k) consists of S_i exponential servers with parameter μ. After receiving service from the ith node, customers proceed to the jth node with probability p_{ij}. Suppose further that the ith node may receive customers from a Poisson stream with rate λ_i from outside the system. Customers at node i depart from the system with probability:

$$q_i = 1 - \sum_{j=1}^{k} p_{ij}$$

That is, the input to the ith node consists of outputs of the other nodes as well as the external input λ_i. The total arrival rates λ_i from outside the system plus the arrival rate from arrivals to node i from (other) internal nodes

$$\sum_j p_{ji}\alpha_j.$$

Medhi (1991) presents Figure 2.7 for the diagram of node i.



where:

$a = \alpha_1 p_{1i}$ $\quad e = \lambda_i$ $\quad h = \alpha_i p_{ij}$

$b = \alpha_2 p_{2i}$ $\quad f = \alpha_i p_{i1}$ $\quad i = \alpha_i p_{ik}$

$c = \alpha_j p_{ji}$ $\quad g = \alpha_i p_{i2}$ $\quad j = \alpha_i q_i$

$d = \alpha_k p_{ki}$

Figure 2.7. Diagram of node i in a Jackson network [from Mehdi, 1991].

Molloy (1989) credits Jackson with determining that even through input processes to a queue are not Poisson, the steady state probability density function for an entire Markov system is the product of the steady state probability density functions for the individual queues. More formally, the joint probability density of the number of customers in each queue in steady state is the product of the marginal probability density functions. Therefore, even though the arrival processes are not Poisson, the random

variables for the number of customers in each queue in steady state are independent. This result has been come to be known as *Jackson's Theorem.*

As part of his theorem, Jackson shows that it is easy to verify that the parameters α_i satisfy the equation:

$$\alpha_i = \lambda_i + \sum_{j=1}^{k} p_{ij}\alpha_j,\ i = 1, 2, \ldots k$$

where α_i is the effective arrival rate to the node i or effective rate of flow through node i.

The major result is that if one properly defines the mean arrival rate at the various centers then the steady state distributions at those centers look exactly like the standard multi-channel systems with which we are familiar (Giffin, 1978). Based on this result, a complex network can be decomposed into a number of simpler subsystems. This means that large networks of queues can be solved by multiplying the results of each queue together (Molloy, 1989). Jackson's theorem shows that for a Jackson network of (Markovian) queues, the particular product-form result of the marginal distribution holds in equilibrium, implying the independence of various nodes in the network (Mehdi, 1991).

2.3.6 Tandem Queues

A common type of queueing network is one in which an arriving part must be serviced by a variety of service distributions before being discharged. Each service center (resource) may provide the input for subsequent centers (Carmichael, 1987). That is, the service facilities are located in sequence. Systems of this type are termed serial, series or tandem queueing systems. This research refers to systems of this type as manufacturing flow lines.

Examples include manufacturing and assembly line operations where a part proceeds through a series of workstations and at each station a different activity is carried out until the completed part passes out of the last station.

Mathematically, tandem queues are the simplest examples of queueing networks. A disadvantage of studying them is that by channeling all parts through a single route may magnify the effects of deviations from such "idealized assumptions as independence and exponential service times" (Wolff, 1989).

A series or tandem queue derives from a sequence of service phases or stages or service stations through which a part passes. The sequence of stages, together with the associated queues, forms the queueing system (Figure 2.8). In this example, arrivals are assumed to occur by a Poisson process with a mean rate λ and service times are assumed to be exponential with mean rates μ_i, i = 1,...,M for an M phase system.

Carmichael p165

Figure 2.8. A series or tandem queueing system [from Carmichael, 1987].

Tandem queueing systems are well studied systems (Table 2.2). Excluding the case when no queues are allowed to form, much of the theoretical development of serial queues has concentrated on the case of exponential interarrival times and service times under steady state conditions. Research has focused on two types of systems. The first is where there is no restrictions on the queue size of any of the stages and the second is

where there is a part or total restriction on the queue size at the stages (Carmichael, 1987).

When there are no restrictions on the queue size for a stage, each stage may be analyzed as a single phase in isolation from the other phases. This analysis may be carried out provided the input to each phase is known and the system input is known and this is Poisson with mean rate λ (Carmichael, 1987).

Table 2.2. Survey of tandem flow line research.

Author	Subject
Altiok (1989)	Develops an approximation for queues in series with phase-type service times and blocking.
Brandwajin and Jow (1988)	Proposes an approximation method for tandem queues with blocking.
Graves (1986)	Develops a discrete-time, continuous-flow model for studying the operation of a job shop that experiences a mix of input job types.
Hillier and Boling (1967)	Explores finite queues in series with exponential or Erlang service times.
Konig and Shmidt (1984)	Develops relationships between the time/customer stationary characteristics of tandem queues attended by a single server.
Ku and Niu (1986)	Studies the stochastic nature of a Johnson's two-machine flow shop with random processing times.
Hendricks (1992)	Explores the output process of a serial production line of exponential machines with finite buffers.
Lee and Zipkin (1992)	Develops an approximation for an exponential tandem queueing system with planned inventories.

Maaloe (1973)	Offers approximation formulae for estimating the waiting-time in multi-channel queueing systems with Poisson arrivals and constant or Erlang-distributed service times.
McCormick, Pinedo, Shenker, and Wolf (1989)	Proposes heuristics to maximize output of an assembly line with blocking.
Pinedo (1982)	Studies the optimization problem of minimizing the completion time in a flow shop.

Table 2.2. cont.

Author	Subject
Shalmon and Kaplan (1984)	Presents a complete analysis of the delays in a tandem network of queues with deterministic service and multiple, interfering sources.
Suresh and Whitt (1990)	Describes the results of a simulation experiment of arranging queues in series.
Wittrock (1988)	Presents an algorithm for scheduling parts through a flexible flow line manufacturing system.
Whitt (1983)	Develops methods for estimating a point process (departure) by a renewal process.
Whitt (1984)	Explores the departure process for a queue with many busy servers.
Wolff (1982)	Explores light traffic results for tandem queues with Poisson arrivals and general service times.

For application purposes, Carmichael (1987) remarks that is it reasonable to treat the stages as independent whenever there is sufficiently large queue capacity allowed at each stage (and the arrival process to a stage can be estimated). Lee (1966) suggest this to be the case when the probability of a customer being turned away is less than 10%.

As soon as restrictions are placed on one or more of the stages' queue sizes, interaction between the phases occurs such that the state of the M-stage system has to be considered as a whole instead of at separate states for each of the stages. In fact, the independence property and its implications no longer hold (Hillier and Lieberman, 1986).

Because of limited queue capacities between stages, blocking occurs. Blocking prevents the smooth movement of parts through stages. Generally, any form of restrictions of queue capacity at any of the stages produces dependency among the phases and all phases have to be considered together in any analysis (Carmichael, 1987).

2.4 The Process of Abstraction

The process of developing an abstract (aggregate) simulation model has two major areas of research. The premise of many conceptual framework for simulation development is a hierarchical, modular structure. These frameworks offer the ability to aid modelers in abstracting a system. Section 2.4.1 reviews simulation conceptual frameworks and concludes that they offer little assistance for this research. Section 2.4.2 explores how aggregation is performed in areas other than simulation.

2.4.1 Simulation Conceptual Frameworks

Law and Kelton (1982) define discrete event simulation to be the modeling of a system as it evolves over time by a representation in which the state variables change only at a countable number of points in time. There are many frameworks for this representation to occur.

This section focuses on conceptual frameworks for simulation models. Pritsker (1986), provides the following insight, "In developing a simulation model, an analyst needs to select a conceptual framework for describing the system to be modeled." This framework or perspective is the "world view" within which the system functional relationships are perceived and described (Pritsker, 1986). Derrick, Balci, and Nance (1989) state that a conceptual framework is the underlying structure and organization of ideas which constitute the outline and basic frame that guide a modeler in representing a system in the form of a model. Conceptual frameworks provide both the implementation and design guidance for the modeler.

Zeigler and Oren (1986) claim that simulation models can be specified in a number of formalisms and simulated (*i.e.,* have their behavior generated) by a variety of methods. Formalisms (conceptual frameworks) are set-theoretic short-hands for specifying a mathematical dynamic system. There is no best formalism to represent the

variety of behaviors in real systems of interest but some formalisms are more natural than others and correspond more directly with a system.

Before a discussion on different world views or conceptual frameworks is presented, a review of key terms and concepts of discrete event simulation is required. *Entities* are the objects within the boundaries of a discrete system. Examples include people, equipment, orders, and parts. The purpose of developing a discrete event model is to monitor the *activities* that the entities engage in so as to analyze the system. In discrete simulation, the state of the system can only change at event times (Pritsker, 1986). A simulation model can be formulated by the following (Pritsker, 1986):

- defining the changes in the state that occur at each event time.
- describing the activities in which the entities in the system engage.
- describing the process through which the entities in the system flow interact.

Figure 2.1 (Pritsker, 1986) depicts the relationship between the concepts of an event, activity, and a process.

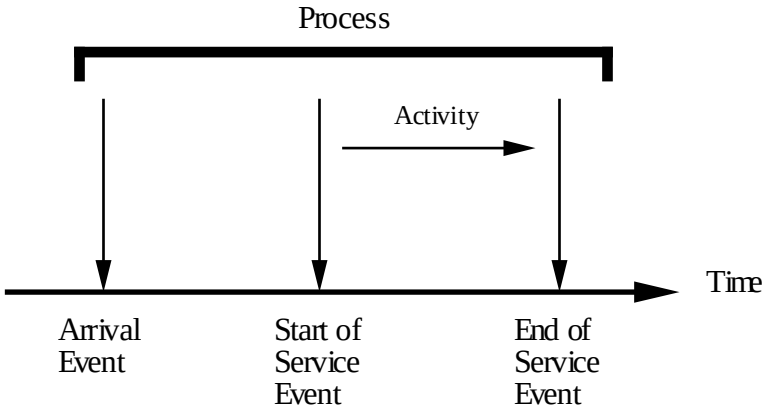


Figure 2.9. Relationship of events, activities, and processes [from Pritsker, 1986].

The scheduling of the next event and the task of updating the system state by the next event increment method can be implemented in several ways (Neelamkavil, 1987). the common event sequencing approaches are Event Scheduling (ES), Activity Scanning

(AS), The Three-Phase Approach (TPA), Process Interaction (PI) and Transaction Flow (TF). These will be first reviewed so as to allow comparisons to recent developments in the area of conceptual framework. These developments have resulted in several additional frameworks: System Theoretic Approach (STA), the Conical Methodology (CM), and the Product Automaton.

Event Scheduling

The event scheduling approach takes a global view of the entire system. With it, a complete description of everything that happens in the model when an individual event occurs is given and the events are scheduled explicitly by specifying their time of occurrence (Neelamkavil, 1987). Hooper (1986) explains that the event scheduling time control procedure selects from the event list the event notice having the earliest occurrence time, updates the simulation clock to that time, and invokes the corresponding event routine. Any testing of conditions, other than on clock time, must occur within specific event routines. Until termination time, events are chosen and processed successively.

Activity Scanning

In activity scanning, no event list is maintained. Rather, the simulation progresses from event to event by scanning activities (Neelamkavil, 1987). Activity scanning chooses the next event based both on scheduled time and condition testing (Hooper, 1986). To achieve this, identification is required for all objects in the system, the activities which the objects perform, and the conditions under which the activities take place. Derrick *et al.* (1989) explain that activity scanning uses a test set of boolean conditions to enable determination of the state change that can initiate an activity. The

test conditions link the various activities together and produce the state transitions of the model objects and the interactions among them (Derrick *et al.*, 1989).

The Three-Phase Method

The three-phase approach is a modification of activity scanning. In this simulation world view, a simulation consists of a number of time dependent scheduled events, plus a number of conditional events that are scanned. O'Keefe (1986) presents the following three step algorithm:

repeat

A: advance time to next scheduled event

B: execute all scheduled events due to occur at this time

C: scan the conditional events

until simulation halted

Activities are classified as B-activities or C-activities. The B-activities are the "bound-to-occur" or "book-keeping activities" that represent the unconditional state changes (unconditional events) which can be scheduled in advance (Derrick *et al.*, 1989). The C-activities are the conditional or cooperative activities that represent the state changes which are conditional upon the co-operation of different objects or satisfaction of specific (compound) conditions.

Process Interaction

The process interaction uses a world view in which components in a system progress through a sequence of steps (called a process). Each step may consist of a condition sequence and an action sequence. Execution of the condition sequence determines whether execution of the action segment should occur (Hooper, 1986). Neelamkavil (1987) discusses that a list of processes, each ordered according to the time of occurrence of the next event, is maintained and the collection of all event sequences together describes all events that occur in the system. Therefore the generation of the next event time and the scheduling of the next event can be achieved indirectly by

activating the process (scheduling and executing the routines which describe the actions of the process) at the head of the list.

Transition Flow

Transaction flow handles time and state relationships similar to the process interaction approach. The main difference, according to Derrick *et al.* (1989), is that in transaction flow, "transactions" are created and moved though blocks, executing specialized actions that are "associated" with each block. The block structure generates a method by which the examination and communication among system components is limited. In addition, as objects (transactions) pass through these blocks, predefined processes are activated which are hidden to the modeler.

System Theoretic Approach

Systems Theory is a scientific discipline whose primary concern is to provide problem solving methods and tools (Rozenblit, 1988). Under the system theoretic approach, a modeler can identify the static and dynamic structure of the model. Derrick *et al.* (1989) explains that this approach is based on set theory. From this the system modeling formalism provides a comprehensive, yet general model representation that allows hierarchical decomposition and abstraction. A system model can be informally represented by describing (Derrick *et al.*, 1989):

- *components* - "the parts from which the model is constructed"
- *descriptive variables* - "tools to describe the conditions of the components at points in time"
- *component interactions* - "the rules by which components exert influence on each other, altering their conditions and so determining the evolution of the model's behavior over time"

The major framework based on the concepts of the Systems Theory is the Discrete Event System Specification (DEVS) developed by Zeigler (1986, 1987). DEVS is a formal specification for discrete event models which use formalism and provides for a variable-time increment time flow mechanism. Derrick *et al.* (1989) states that DEVS provides the static structure of the model. The model dynamic structure is obtained though how components interact.

DEVS provides a means of specifying a mathematical object called a system. A system is composed of a time base, inputs, states, outputs, and functions for determining the next states and outputs given current states and inputs (Rozenblit and Jankowski, 1991).

In DEVS formalism, basic atomic models are defined by the following structure:

$$M = < X, S, Y, \delta int, \delta ext, \lambda, \tau >$$

where:

X is the set of external input event types

S is the sequence state set

Y is the set of external even types generated as output

δint (δext) is the internal (external) transient function dictating transient due to internal (external input) events

λ is the output function generating external events as the output

τ is the time advance function

With DEVS, a model must be viewed as possessing input and output ports through which all interaction with the environment is mediated. Rozenblit and Jankowski (1991) state that a basic model contains the following information.

- the set of input ports through which external events are received
- the set of output ports through which external events are sent
- the set of variables and parameters

- the time advance function which controls the timing of internal transitions
- the internal transition function which specifies to which next state the system will transmit after the time given by the time advance function has elapsed
- the external transition function which specifies how the system changes when an input is received; the next state is computed on the basis of the present state, the input port and value of the external event, and the time that has elapsed in the current state.
- the output function which generates an external output just before an internal transition takes place.

Basic or atomic models may be coupled to form a multi-component model. In DEVS, these are referred to as coupled models. This approach allows for the development of hierarchical model construction. The development of the hierarchy allows for a component in the composition tree to be an atomic model or a coupled model. When a coupled model is used, it is constructed from one or more components. Figure 2.10 illustrates the general recursive pattern for model construction.

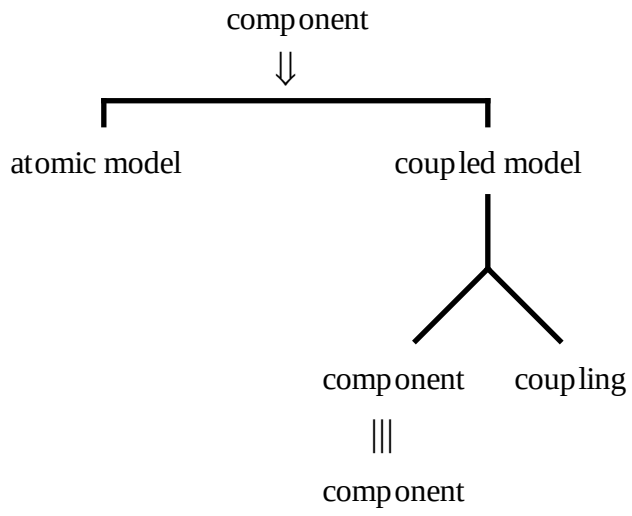


Figure 2.10. Recursive hierarchical model construction with DEVS [from Zeigler, 1986]

The development of modular discrete event models using this framework requires looking at model differently. A model must be viewed as processing input and output ports through which all interactions with the environment are controlled. Events determine the values appearing at the ports. Appearances occur when external events (those arising outside the model) are received on input ports. Internal events (those arising within the model) change the state and are themselves events which need to be transmitted to other model components. The goal of the model is to decide how to respond to these event.

Product Automaton

The Product Automaton formalism, proposed by Portier (1987), is a formalism for discrete simulation that allows one to specify a discrete simulation model in "a precise and unambiguous manner." The approach of the product automaton is similar to DEVS in that it uses modularity and a hierarchy. In addition, both define a system as a finite decomposition of subsystems which are maintained in a modular fashion (Portier, 1987). The differences arise in their perceptions of the world view, the definition of state, and state transitions.

The product automaton can be described as a mathematical object. Let T be a rooted tree, then $T \equiv <N, E, r>$ where N is a finite set called the node set, E is a subset of N x N called the edge set and the node $r \in N$ is called the root and <N, E> is a tree.

A product automaton is a structure

$$PA \equiv <N, E, r, \{M_i\}, \{Z_{i,j}\}>$$

where <N, E, r> is a rooted graph

$$M_i \equiv <X_i, S_i, \partial_i, ta_i> \text{ for all } i \in N$$

Each M_i is called a component. Each component is made up of the following.

X_i is a finites set, the inputs to i

S_i is a set, the state set of i

∂_i is a function, the state transition function of i

$ta_i \in R^+_{0,} \square$ is the natural update time of i

The product automaton framework is a collection of interacting and interrelated components. The components are arranged in a decomposition tree and interact only with adjacent components as defined by the tree (Portier, 1987). To make the relationship between the components more complete a factor is defined as part of some decomposition, and a product is a component that has many factors.

Similar to how Zeigler defines a system as an atomic model or a coupled model, Portier defines a system to be either atomic or made up of subordinate subsystems (*i.e.* coupled). As part of the formalism, every component i maintains a current state (an element of S_i), accepts inputs (from X_i), and changes state (using ∂_i). An atom has state transitions based only on its received input and on its internal state whereas a product changes based on the state of its factors.

Conical Methodology

The conical methodology provides the "fundamental requirements that underlie a model development system" (Overstreet and Nance, 1985). The objective of a model development systems is to provide tools to reduce the cost of constructing simulation experiments while improving the quality of the information those experiments produce. Overstreet and Nance states that "the CM (conical methodology) provides a carefully structured approach for documented, effective model specification construction."

The conical methodology, developed for simulation modeling tasks, uses a top-down model definition followed by a bottom-up model specification (Balci and Nance, 1987). Top-down model definition produces a static model representation and is

accomplished through a hierarchical decomposition of the model into successive submodels. This is accomplished by requiring the modeler to perform an object decomposition, assigning attributes to objects based on the system being described and the objectives of the simulation study. At each level of decomposition, attributes are assigned and classified by type (Derrick, Balci, and Nance, 1989).

The model-specification phase (bottom-up specification) uses the static representation to produce a dynamic representation. Specification is started at a base-level submodel in the decomposition hierarchy and is performed successively at higher levels until the model level is reached.

Relationship Between This Research and Conceptual Formalisms

Conceptual frameworks provide two types of guidance. First is the implementation guidance (algorithmic, managerial supervisory) which directly impacts the subsequent executable form of any model. Conceptual frameworks also provide design (structural, existential, skeletal) guidance (Derrick *et al.,* 1989). With this, the modeler is assisted in the development of the static and dynamic model structure for identifying the objects, their attributes, and their rules of interaction.

Conceptual frameworks other than the classical ones provide design guidance. They are used in the development of the static and dynamic structure of the model. This research focuses on developing an aggregate model of a system description. Within this scope, emphasis is on identify how to lump or combine system components in the development of the conceptual model.

The conceptual model and the conceptual frame are not the same (Figure 2.11). The model is the “picture” developed by the modeler of the necessary components, features and their associated interaction. The conceptual frame (world view) is the

underlying structure and organization of this conceptual model which is used to develop the simulation model.

The conceptual framework identifies and builds a model of those features identified in the development of the conceptual model. The relationship between the two is analogous to the relationship between flour and cake. The conceptual model is the flour and the conceptual frame (or world view) is the cake. Thus, the conceptual model is an ingredient (the primary ingredients) which makes for the conceptual framework. But to make a cake (conceptual framework) other ingredients and items are required. For instance, a pan is needed.

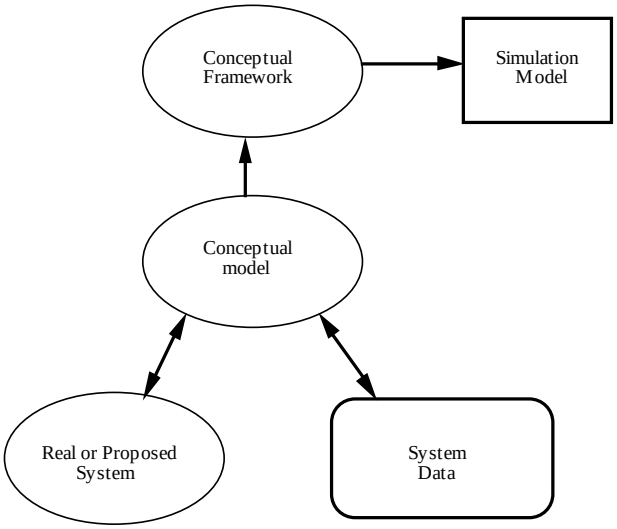


Figure 2.11. Relationship between the conceptual model and the conceptual framework.

Several of the reviewed conceptual frameworks are based on the premise of hierarchical, modular development with decomposition. A conceptual framework is only the "skeleton" which a modeler uses for creating a model. It can be argued that the conceptual framework, by providing a structure for aggregation assists the modeler. Unfortunately, the conceptual frameworks do not provide the means to identify a correct hierarchy or module. Thus, the task of developing modules and forming them into a hierarchy is based on a modelers ability. For instance, does a cake have any influence on

how the flour was produced? That is, does the conceptual framework have any influence on how the conceptual model is specified? It can be reasonably assumed that the conceptual framework (cake) has no influence in the development of the conceptual model (flour).

To develop a conceptual framework using any of these formalisms requires the modeler to develop the system entity structure by deciding on what entities exist in the system, the relationship between them. In short, development of the conceptual model is the modeler's task. Conceptual formalisms offer the "shell" for storing the model information, but do not offer techniques for aiding in the abstracting (aggregating) the system.

2.4.2 Aggregation Research

Formally, abstraction is the technique of reducing a system description to a level of detail which can more easily be managed. One of the primary abstraction techniques is aggregation. This technique involves combining or lumping details into a single, equivalent function. As indicated in Chapter 1, many authors claim that developing an abstract (aggregate) simulation model is an "art". This belief among simulation researchers and practitioners is prevalent since there are no formal techniques for developing an abstract (aggregate) simulation model.

The complexity of a manufacturing system is determined largely by the number of resources explicitly modeled (Dietrich, 1991). Rogers *et al.* (1991) remarks that, "A fundamental issue in the use of optimization models is the tradeoff between the level of detail and the ease of solving the model." Aggregation and disaggregation techniques have proven a valuable tool for manipulating data and determining the appropriate policies for this tradeoff. Rogers *et al.* (1991) develops a general framework for

aggregation and disaggregation methodology, and explore its potential for solving optimization problems such as linear programming problems and network flow problems.

In developing a simulation model, most descriptions state that it is the responsibility of the modeler to "model at an appropriate level of detail". Unfortunately, there are no formal guidelines or rules to follow. Seeing this as a problem, Antonelli *et al.* (1986) developed a useful tool for the development of flexible automation. This tool is a system description language which can generate a complete functional description of a manufacturing cell of arbitrary complexity. It proposes a description system based on the concept of hierarchical decomposition utilizing the Ada programming language in conjunction with established diagramming decomposition methods. The distinguishing aspect of this work is that is takes advantage of certain features of Ada (such as type checking) to create a description that can be automatically verified for consistency.

One of the few papers relating to this research is a paper that Henry Friedman published in 1965 on reduction methods for tandem queuing systems. He developed a reduction procedure based on the dominance of a queue's impact on the other queues of the flow line. Applying his procedure results in modeling only the dominant queues of the system. The other, less dominant, queues are represented by only their service means.

Gershwin (1987) presents a decomposition method for evaluating performance measures of tandem queueing systems with finite buffers in which blocking and starvation are important. In general, these system are difficult to evaluate because they have large state spaces and cannot be decomposed exactly. The procedure works by approximating a single k-machine flow line by a set of k-1 two-machine lines. Performance measures are developed for the two-machine lines and combined to form estimates of the overall system. Expanding on this work, Gershwin (1989) refines his original algorithm to consider the case of unreliable tandem queueing systems. Takahashi

(1989) offers a similar procedure which works by looking at a node at a time, two adjacent nodes at a time, three adjacent nodes at a time, and so on.

Schweitzer and Altiok (1989) develop an aggregation procedure of modeling tandem queues without intermediate buffers. Their work is limited to the case of exponential service times. One of the underlying premises is the belief that aggregation permits a system to be reduced to (say) one server at a time or (say) two servers at a time.

A subset of other authors researching the concept of approximate decomposition of tandem queueing model include: Hunt (1956), Hillier and Boling (1966), Takahashi, Miyahara, and Hasegawa (1980), Altiok (1982), Gun and Makowski (1989). The majority of these efforts focus on exponential queueing systems with finite buffers.

2.5 Chapter Summary

The objective of this chapter was to provide a foundation for which the aggregation methodology for creating an aggregate simulation model of a manufacturing flow line can be developed. To achieve this, this chapter reviews:

(1) *Simulation*. After the definition of a model is presented, the concept of a computer simulation model is discussed. The components and steps of a model's development are reviewed. In addition, the important task of how to generate random variable with simulation is summarized. This will be key to understanding the final step of the aggregation methodology presented in the next chapter.

(2) *Discrete Manufacturing System*. Since this research is concerned with production flow line systems, general discrete manufacturing systems are defined and a detailed discussion of flow line manufacturing systems is given. In addition, common performance measures for a manufacturing simulation study are reviewed.

(3) *Queueing Theory*. As the next chapter will illustrate, the core techniques of the aggregation methodology are based on queueing theory. An introduction to queueing and queueing estimates for the M/M/S, M/G/S, and G/G/S queueing systems are given. In addition, the concept of a queueing networks are reviewed. As a special case of a queueing network, tandem queueing systems are studied.

(4) *Aggregation*. Conceptual simulation frameworks were extensively reviewed with the hope that they would offer techniques for how to develop an abstract (aggregate) simulation model. Though several of them provide the means for aggregation to occur, they offer no specific rules or procedures for accomplishing it. In addition, a review of relevant aggregation research from different areas is studied.

The presentation of these topics establishes the groundwork for the next chapter to present the methodology for creating an aggregate simulation model of a flow line manufacturing system.

CHAPTER 3

RESEARCH METHODOLOGY

In developing a simulation model, most of the actual features of the system under study must be ignored and an abstraction must be developed. If done correctly, this idealization provides a useful approximation of the real system. Aggregation is one of the available techniques for abstracting a system. It involves aggregating or lumping details into approximately equivalent functions (Pegden *et al.*, 1990). Potential benefits of developing an aggregate simulation model include a reduced run length, a less complex model, and less demand of system resources.

The objective of this research is to develop a formal methodology for creating an aggregate simulation model that can be used to estimate average part cycle time. The methodology operates by aggregating or lumping together resources of the system into equivalent aggregation resources. Determining the specifications for representing these aggregation resources in an aggregate simulation model is the key task of this research.

Developing a simulation model is an iterative process. Developing an aggregate simulation model of a manufacturing systems follows a similar evolution. The chapter first discusses what a flow line manufacturing system is and develops a formalism for specify the data of such a system. From this description, summary statistics are computed. Using these results, the simulation data is combined such that an aggregate simulation model can be developed. A pictorial view of the steps of the aggregation procedure are described in Figure 3.1.

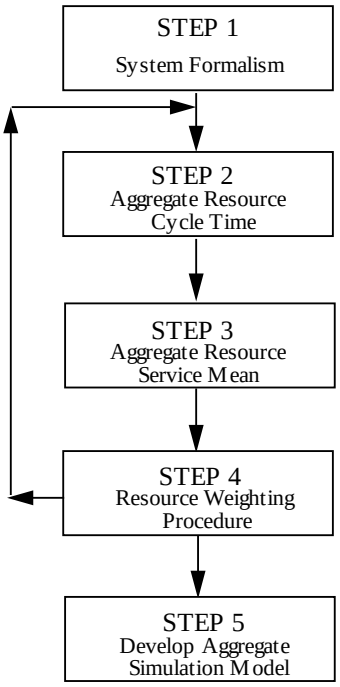


Figure 3.1. Steps of the aggregation methodology.

The remainder of this chapter will be devoted to addressing each of these steps. Issues relating to the purpose, development, and application of each will be presented.

3.1 System Formalisms

A necessary task in any simulation project is to collect or specify data on the system under study. This section provides a formalism for collecting and representing the information of a manufacturing flow line. Section 3.1.1 defines a flow line manufacturing system and presents a formalism for representing it. Section 3.1.2 introduces the manufacturing flow line of MPD manufacturing company. To illustrate the application of the methodology, this example flow line will be continued throughout this chapter. Section 3.1.3 mirrors 3.1.1 in that a formalism is developed for representing the aggregation of a flow line system. Section 3.1.4 continues the MPD manufacturing system example by developing its aggregate representation.

3.1.1 Flow Line Formalism

Pinedo (1982) defines a flow line (flow shop) system as a manufacturing facility in which there are *n* machines and *m* job types to be processed. The *m* parts or jobs are processed by the same *n* machines with the ordering of processing at different machines being the same for all jobs. Hence, each part has to be processed first on machine 1, then on machine 2, etc.

Unfortunately, Pinedo's definition is both too broad and too narrow for certain aspects of this research. It exceeds this research in that it allows for *m* part types to flow through the manufacturing system. This research assumes that there is only *one* part type in the system. Pinedo's definition is to restrictive in that it states that parts go from machine to machine in a sequence. While this is technically true, it in not entirely accurate, for parts go from a machine to a queue or buffer area, and then to the next machine.

The need for these adjustments will become more apparent as the methodology is discussed later in this chapter. To adjust for these problems, consider the following modification to Pinedo's (1982) definition: *The single part type is processed at N production stations (resource) with the ordering of processing at a production station (resources) being the same for all parts*. The phrase "production station" replaces machines and "single part type" has been added. Thus, a flow line is a sequence of N production steps or resources (R_i), consisting of a machine and associated buffer (or queue) area. This relationship is illustrated in Figure 3.2. A description of the basic notation and definitions are given in Table 3.1.

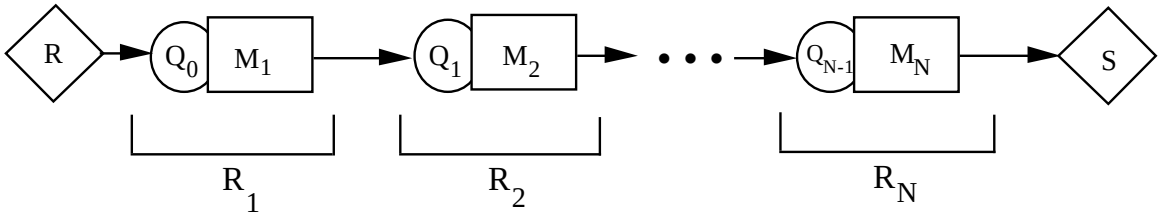


Figure 3.2. A flow line consisting of N production steps, where each resource or production step consists of a machine and associated waiting or buffer area.

Table 3.1. Description of flow line notation.

R	Receiving area
S	Shipping area
N	Number of production steps to produce a part
R_i	A resource or production step consisting of a queue and associated machine (i = 1 to N)
M_i	Machine i (i = 1 to N)
Q_j	Queue or buffer proceeding M_{j+1} (j = 0 to N-1). For simplicity, Q_0 is represented as a separate queue when in fact it is part of R, the receiving area.

The basic assumptions associated with the flow line production system that this research explores are summarized in Table 3.2 (based on Hendricks, 1992). The most important assumptions are that the time between which parts arrive to the service area follows an exponential arrival and that the queues or buffers between the queues have infinite capacity.

Table 3.2. Basic assumptions of manufacturing flow line [based on Hendricks, 1992].

1. The production line (flow line) is a series arrangement of a finite number of N resources. The machine component of a resource has s_i parrallel servers and each server can operate on one part at a time and has internal storage for that part.
2. The production line is operating under steady state conditions.
3. The machines M_i ($i = 1,...,N$) have mutually independent processing times. The coefficient of variation of the service time is required to be less than or equal to one.
4. Parts leave the receiving area (arrive to Q_0) following an exponential distribution with density function $f(t) = \lambda \exp(-\lambda t)$.
5. The shipping area has unlimited storage capacity and the receiving area has an unlimited supply of parts.
6. All machines are reliable and produce no bad (or scrap) parts.
7. No batching and no setup times are allowed.
8. All queues between machines have infinite storage capacity.
9. The flow line does not allow for feedback or rework.

From this description and listing of assumptions, a more symbolic relationship can be used to describe the flow system. This description is used to collect all the necessary data for developing the aggregate representation of the system. Table 3.3 provides the formalism which symbolically describes a manufacturing flow line. Items within angular brackets (*i.e.*, < >) are information that must be defined by the user or will be specified by the aggregation methodology.

Table 3.3. Flow line formalism.

$$FL = \langle R, R_1, \ldots, R_N, S \rangle$$
$$R = \langle 1/\lambda, Z \rangle$$
$$S = \langle U \rangle$$
$$R_i = \langle Q_{i-1}, M_i \rangle \quad i = 1, \ldots, N$$
$$Q_{i-1} = \langle v_{i-1}, x_{i-1} \rangle \quad i = 1, \ldots, N$$
$$M_i = \langle F_i, m_i, s_i \rangle \quad i = 1, \ldots, N$$

A flow line (FL) consists of three primary components, the receiving area (R), the shipping area (S), and N production steps (R_i). The receiving area (R) is described by the mean time between arrivals ($1/\lambda$), where λ is the arrival rate, and Z, which is the maximum number of parts that can arrive from the storage area. The shipping area (S) is characterized by its storage capacity (U). From the assumptions of Table 3.2, λ follows an exponential distribution and Z and U are assumed to be infinite.

Each production step or resource (R_i) is composed of a queue (Q_{i-1}) and a machine (M_i) which is to service (process, inspect or machine) a part. The queue component of a resource represents the waiting space preceding the machine on which a part waits until a server becomes available to process it. It is characterized by its buffer capacity (x_{i-1}) and the variability between arrivals to the queue. Table 3.2 indicates that this research assumes that there is infinite queue or buffer capacity (*i.e.*, $x_{i-1} = \infty$). The variability between arrivals (v_{i-1}) to the buffer is important for applying the aggregation techniques of this research. This value is not specified, but rather it will be calculated with a procedure that will be discussed in Chapter 4.

Each machine (M_i) is specified by its service time distribution (F_i) and service mean (m_i) to service a single part. For example, this characterization might be: uniform with a minimum of 10 and a maximum of 20 such that the mean is 15. The only

restriction placed on these values is that the coefficient of variation (standard deviation divided by the mean) of the service mean is less than or equal to one. The need for this restriction will be discussed in Chapter 4. Though this requirement is restrictive, it does allow for the use of all common simulation distributions (*e.g.*, triangular, uniform, normal, exponential, Erlang, beta, and Weibell).

A machine is also characterized by the number of parrallel, identical servers that perform the machine's task. This research assumes that the number of servers (S_i) is greater that or equal to one (*i.e.*, $s_i \geq 1$).

3.1.2 MPD Flow Line

To illustrate the use of the formalism in describing a production flow line system, consider the following example from company MPD that will be used throughout this chapter. The picture depicting MPD's system is presented in Figure 3.3. Table 3.4 provides the mathematical formalism that describes this system. It should be understood that the MPD example is only an illustration and does not represent a true manufacturing system. It has been artificially designed to illustrate all the steps required of the aggregation methodology. More detailed examples will be presented in Chapter 4.

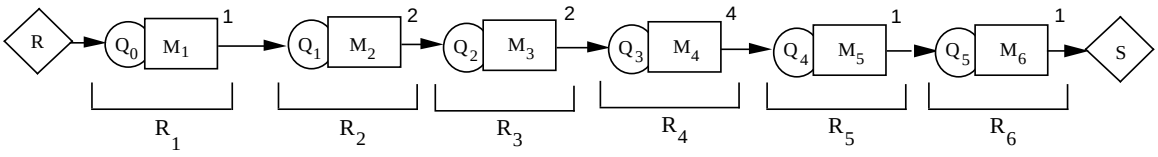


Figure 3.3. MPD manufacturing flow line. This system consists of 6 resources, where R_1, R_5, and R_6 each have 1 servers, R_2 and R_3 each have 2 servers, and R_4 has 4 servers.

Table 3.4. Flow description of MPD's system. Distributions are specified for the arrival and service time of parts.

$$FL = \langle R, R_1, R_2, R_3, R_4, R_5, R_6, S \rangle$$

$$R = \langle 100, \infty \rangle$$

$$S = \langle \infty \rangle$$

$R_1 = \langle Q_O, M_1 \rangle$	$R_2 = \langle Q_1, M_2 \rangle$	$R_3 = \langle Q_2, M_3 \rangle$
$Q_O = \langle v_O, \infty \rangle$	$Q_1 = \langle v_1, \infty \rangle$	$Q_2 = \langle v_2, \infty \rangle$
$M_1 = \langle Uniform(75,85), 80, 1 \rangle$	$M_2 = \langle Normal(130,15), 130, 2 \rangle$	$M_3 = \langle Triangular(120,150,180), 150, 2 \rangle$
$R_4 = \langle Q_3, M_4 \rangle$	$R_5 = \langle Q_4, M_5 \rangle$	$R_6 = \langle Q_5, M_6 \rangle$
$Q_3 = \langle v_3, \infty \rangle$	$Q_4 = \langle v_4, \infty \rangle$	$Q_5 = \langle v_5, \infty \rangle$
$M_4 = \langle Normal(320,25), 320, 4 \rangle$	$M_5 = \langle Triangular(32,43,60), 45, 1 \rangle$	$M_6 = \langle Uniform(64,80), 72, 1 \rangle$

The MPD flow line system consists of six resources, where resources R_1, R_5, and R_6 each have single server machines, R_2 and R_3 each have two servers, and R_4 has four parallel, identical servers. Parts arrive to the flow line following an exponential distribution with a mean time between arrivals of 100 minutes. Each of the resource queues is characterized by its arrival variability (to be discussed in Chapter 4) and its storage capacity (assumed to be infinite). Each machine is summarized by its service distribution (*e.g.*, Normal with a mean of 130 and a standard deviation of 15), its mean service time, and its service capacity (1, 2, or 4 in this example). For instance, the service distribution of resource R_3 is the triangular distribution with parameter values of 120, 150 and 180 as the minimum, mode, and maximum, respectively. In addition, it has a mean service time of 150 and two parallel, identical servers for performing the production task.

3.1.3 Aggregate Formalism

The formalism presented previously describes the “as is” flow line manufacturing system. At present, it offers no means to account for the aggregation of manufacturing steps in the production sequence of a part.

This research proposes that in an aggregation representation of a system, all resources with a given server capacity are aggregated together to form a new aggregate resource, AR_i, where i represents the aggregate resource service capacity. For example, AR_1, represents all single server resources from the original system and AR_2 represents all two server resources.

Obviously many other characteristics of a flow line system could have been used as the aggregation feature. Examples include the type of work performed by the resource (machining, inspection, assembly, etc.), the service time distributions, the coefficient of variation, or even the utilization level of a resource. The decision to aggregate on the service capacity of a resource is based on the fact that this is the one characteristic that remains constant for any type of flow line system. For instance, all flow line systems have a set numbers of servers for each resource, whereas they do not have all assembly or inspection stations. Thus, to make the aggregation approach general to the widest type of flow line system, resources are aggregated together based on the number of parrallel, identical servers performing that resource’s operation.

Figure 3.4 provides a possible pictorial representation of an aggregated system. As with the original system (Figure 3.2), parts arrive to a receiving area and exit the system by a shipping area. The figure demonstrates that a part is processed through a series of aggregated resources (AR_i), where each represents the aggregation of all i server capacity resources from the original flow line.

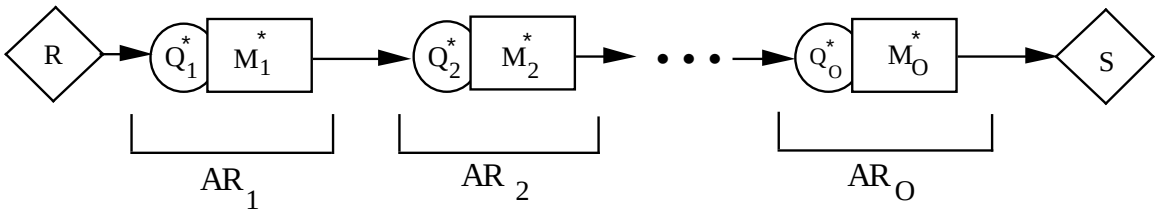


Figure 3.4. Aggregate flow line representation of a manufacturing system. Resources are aggregated according to their service capacity. The original system is aggregated into a total of O aggregated resources, where O is the maximum service capacity of all resource in the original system.

The representation demonstrated in Figure 3.4 indicates that a part first proceeds to queue Q_1^* where is waits to be processed on machine M_1^* which has a single server. After processing, a part next waits in queue Q_2^* for service on machine M_2^* which has two servers. This continues until the part is processed by all the O aggregation resource. Note that the representation of how aggregation resources are positioned or ordered is but one of the many possible combinations. This research assumes (see Chapter 1) that resources act independently of one another in estimating the cycle time of part. Thus, Figure 3.4 would be valid if the flow was AR_O to AR_2 to AR_1. The ordering is not important since the objective of this research is to aggregate in order to estimate the average cycle time of a part.

As such, Figure 3.5 is a more accurate description of how aggregate resources will be modeled in the aggregate simulation model. In this representation, resources are modeled independently of one another, such that the order is insignificant. As the figure indicates, when a part arrives to the aggregate flow line (following an exponential distribution), it is sent to each of the aggregation resources. Thus, order is removed, and the arrival distribution of a part to any aggregate resource can be assumed to be exponential. This procedure follows the work of Kleinrock (1976). He showed that for Poisson arrivals, general service, and FIFO queues, one can approximate each station in a tandem flow line system as an M/G/S queue. Otherwise, the arrival process to each

station (aggregation resource) must be determined. Chapter 4 will further discuss this case.

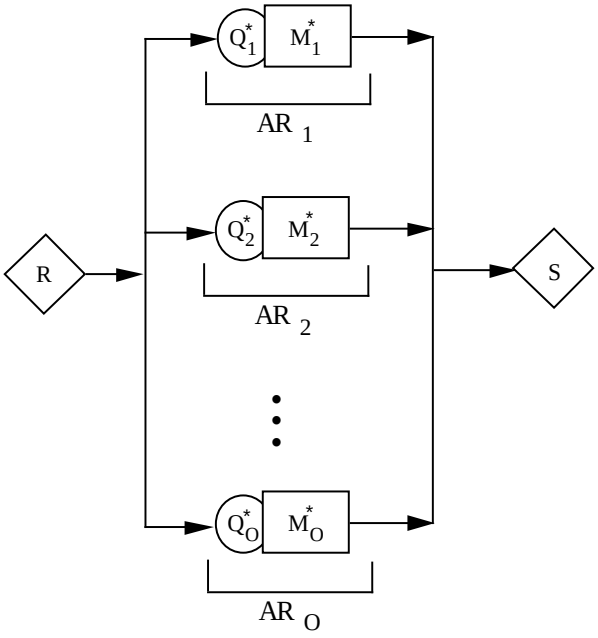


Figure 3.5. Representation of an aggregate flow line to estimate cycle time.

As with the original system description, the abstract system can be described with a symbolic formalism (Table 3.5). Items within angular brackets are those that must be defined by the user. Curly brackets (*i.e.*,{ }) indicate that of the items within the brackets, only one should be selected or specified.

Table 3.5. Aggregate flow line formalism.

$$AFL = \langle R, AR_1, \ldots, AR_O, S \rangle$$
$$AR_i = \{\varnothing, \langle Q_i^*, M_i^* \rangle\} \quad i = 1,\ldots,O$$
$$Q_i^* = \langle x_i^* \rangle \quad i = 1,\ldots,O$$
$$M_i^* = \langle F_i^*, \delta_i^* \rangle \quad i = 1,\ldots,O$$

An aggregated flow line consists of a receiving area (R), a shipping area (S), and a collection of O aggregation resources (AR_i), where O is the maximum number of parallel,

identical servers used by any machine in the flow line. For those aggregation resources that exist (*i.e.*, not the empty set), they are characterized by a queue and associated machine with server capacity i.

The queue (Q_i^*) component of an aggregation resource is defined by its storage capacity. As with the original system, the buffer capacity is assumed to be infinite (*i.e.*, $x_i = \infty$). The machine, M_i^*, represents all the machines of the original system with capacity i. That is, $M_i^* = \{M_j : S_j = i\}_{j=1,\ldots,N}^{i=1,\ldots,O}$. A machine in an aggregation resource is characterized by its service distribution (F_i^*) and its average service mean (δ_i^*). That is, F_i^* represents the combined distribution for all the aggregated resource service times for each of the original machines and δ_i^* is the average service mean. Developing the procedure to estimate these values from the original system resource distribution (F_i) is the objective of this research.

3.1.4 MPD Aggregation Flow Line

Figure 3.6 presents the aggregate description of the MPD flow line system. In the aggregated representation of the system, resources R_1, R_5, and R_6 (each with one server) are aggregated to form aggregation resource AR_1 (aggregation of all single server resources), which is represented by queue Q_1^* and machine M_1^*. In addition, resource R_2 and R_3 have been aggregated to form AR_2. Aggregate resource AR_3 is defined as the empty set, since there are no three server resources in the original system. As such, it is not explicitly modeled. R_4 is the only four server resource in the original system, thus it has nothing to aggregate with and is directly mapped to AR_4.

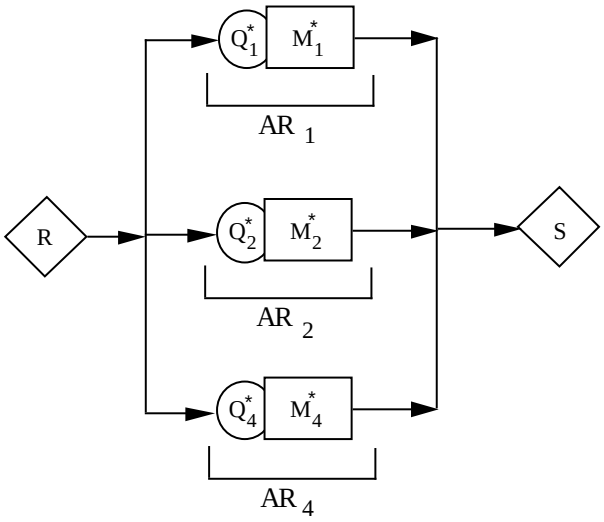


Figure 3.6. Aggregation of MPD's flow line system. AR_3 is not represented since no three server resources exist in the system.

The aggregate flow line formalism of MPD's system is presented in Table 3.6. Note that the aggregate resource service distributions F_1^*, F_2^*, and F_4^* are not specified. The reason for this is that they are not known at this time. It is the objective of this aggregation methodology to develop a procedure so as to estimate them.

Table 3.6. Aggregate flow line formalism of MPD's system.

$$AFL = \langle R, AR_1, AR_2, AR_3, AR_4, S \rangle$$

$AR_1 = \langle Q_1^*, M_1^* \rangle$	$AR_2 = \langle Q_2^*, M_2 \rangle$	$AR_3 = \varnothing$	$AR_4 = \langle Q_4^*, M_3 \rangle$
$Q_1^* = \langle \infty \rangle$	$Q_2^* = \langle \infty \rangle$		$Q_4^* = \langle \infty \rangle$
$M_1^* = \langle F_1^*, \delta_1^* \rangle$	$M_2^* = \langle F_2^*, \delta_2^* \rangle$		$M_4^* = \langle F_4^*, \delta_4^* \rangle$

3.2 Cycle Time of an Aggregation Resource

The previous section defined the manufacturing flow line and its equivalent aggregate representation. It showed how resources of a flow line system are aggregated into aggregation resources, where each aggregation resource represents all resources with the same capacity from the flow line system. For instance, aggregation resource AR_3 represents the aggregation of all three server resources of the flow line. Section 3.2.1 summarizes a technique for determining the average cycle time of an aggregation resource ($\overline{T_i^*}$). Determining this key value is the second step in developing the procedure for developing an aggregate simulation model. Section 3.2.2 applies the procedures of Section 3.2.1 to the MPD manufacturing example.

3.2.1 Computing Cycle Time

The first step in determining the average cycle time of each of the O aggregation resource is to compute the expected cycle time (T_j) of all N resources in the flow line. That is, determine the total steady state processing/service and waiting time (also refereed to as the sojourn time) that a part will experience at each of the resources. By applying the assumption that the cycle time of a resource is independent of one another, the order of resources in the flow line does not matter. The case when order does impact the analysis will be further discussed in Chapters 4. Combining the independence assumption with the fact that parts arrive to the flow in a Poisson fashion, the time between part arrival to any resource can be modeled with an exponential distribution. In actuality, the only resource with a true Poisson arrival is the first resource of the flow line (except for the case of resources having exponential service time). Applying the independence assumption means that any of the resource could be first resource of the flow line. Since parts arrive following an exponential distribution, the arrival to any

resource can be modeled with the exponential distribution (since it could be the first in the flow line).

With the arrival of parts to any resource assumed to be Poisson, the time between arrivals can be modeled with an exponential distribution. In the case when a resource service time distribution is exponential, determining the cycle would allow the use of standard M/M/S queuing formulas, since the arrival and service distributions are both exponential. Correspondingly, if a resource has a general service distribution (*e.g.*, Normal, Uniform) and the service capacity is one, then M/G/1 queueing results can be used.

This research allows for exponential arrivals, general service time distributions, and unlimited server capacity, thus M/G/S queueing formulas must be used for computing the cycle time. To provide general results for the widest set of cases, the M/G/S formula presented in Chapter 2 will be used. The advantage of this formula is that it is relatively simple to use and it is exact for the M/M/S and M/G/1 queuing systems. Rewriting this formula in terms of the flow line terminology results in the following estimator of a resource's cycle time:

$$E[T_j] = \frac{1 + cv_{m_i}^2}{2\lambda(1-\rho_j)}\rho_j C_j + m_j \quad j = 1,\ldots,N$$

where: $E[T_j]$ Expected cycle time of resource j (j = 1,...,N)

$cv_{m_j}^2$ Squared coefficient of variation of the service time of resource j (j = 1,...,N)

λ Arrival rate of parts to the flow line

ρ_j Traffic intensity of resource j (j = 1,...,N): $\rho_j = \frac{\lambda m_j}{s_j}$ j (j = 1,...,N)

s_j Number of parallel, identical servers for resource j (j = 1,...,N)

m_j Mean service time of resource j (j = 1,...,N)

C_j The probability that a part arriving to resource j (j = 1,...,N) has to wait for service: $C_j = \frac{(\lambda m_j)^{s_j}}{s_j!(1-\rho_j)} P_{0_j}$

P_{0J} The probability that zero service are busy for resource j (j = 1,...N):

$$P_{0_j} = \frac{1}{\left[\sum_{n=0}^{s_j-1} \frac{(\lambda m_j)^n}{n!} + \frac{(\lambda m_j)^{s_j}}{s_j!(1-\lambda m_j / s_j)}\right]}.$$

Apply this formula allows one to compute a resource's cycle time. Once values have been computed for all N resource, the next step is to determine the average cycle time of the O aggregation resources in the aggregate flow line system. Applying the assumption that all resources are independent of each other, determining the average cycle time of a aggregation resource reduces to determining the individual cycle times for all the resources represented by an aggregation resource and then summing the cycle time (T_j) of the P_i resource aggregated by AR_i to obtain the total aggregate cycle time (T_i^*). That is,

$$T_i^* = \sum_{R_j \in AR_i} T_j \quad \begin{matrix} i = 1,\ldots,O \\ j = 1,\ldots,N \end{matrix}.$$

But an aggregation resource is defined to be the average of all the resources it represents. Thus, the average cycle time of an aggregation resource is defined as:

$$\overline{T_i^*} = \frac{T_i^*}{P_i} \quad i = 1,\ldots,O$$

That is, the average cycle time of an aggregation resource is the sum of all resource cycle time aggregated by the aggregation resource divided by the number of resources aggregated.

3.2.2 Cycle Times for MPD Example

To illustrate the concepts of this chapter, this section continues the example for MPD's manufacturing flow line. In this example, there are six resources (R_1,...,R_6) in the

original flow line. While the previous section defined the system, this section seeks to determine the average cycle time for each of the three aggregate resources of the aggregate flow line.

Table 3.7 presents the results of computing summary statistics of the resources. For instance resource R_1 has one server, a mean service time of 80 minutes, and a service time variance of 8.3333 minutes^2. As such, the squared coefficient of variation is computed to be .00130208. Applying the techniques of this section (the M/G/S queuing formula) results in R_1 having an estimated cycle time of 240.208 minutes. That is, on average, a part will spend 240.208 minutes waiting for service and being service by R_1. Correspondingly, the cycle times for the other resources are: R_2 (178.187 minutes), R_3 (247.071 minutes), R_4 (440.015 minutes), R_5 (63.7106 minutes), and R_6 (164.952 minutes).

Table 3.7. Summary statistics for the resources of the MPD flow line system. The variance, squared coefficient of variation, and cycle time has been computed for each of the resources.

	R_1	**R_2**	**R_3**
Servers (s_j) =	1	2	2
Mean (m_j) =	80	130	150
Variance ($\sigma^2_{m_j}$) =	8.3333	225	150
SQ. C of V ($cv^2_{m_j}$) =	.00130208	.0133136	.00666667
Cycle Time (T_j) =	240.208	178.187	247.071

	R_4	**R_5**	**R_6**
Servers (s_j) =	4	1	1
Mean (m_j) =	320	45	72
Variance ($\sigma^2_{m_j}$) =	625	33.1667	21.3333
SQ. C of V ($cv^2_{m_j}$) =	.00610352	.0163786	.00411523
Cycle Time (T_j) =	440.015	63.7106	164.952

With the six resource cycle times computed, the next task is to compute the average cycle time for an aggregation resource. For illustration, consider the case of AR_1. This aggregation resource represents all single server resources ($AR_1 = \{R_1, R_2, R_5\}$). The total cycle time for AR_1, T_1^*, is the sum of the cycle time of all the resources it represents. Therefore,

$$\begin{aligned} T_1^* &= T_1 + T_5 + T_6 \\ &= 240.208 + 63.7106 + 164.952 . \\ &= 468.8706 \end{aligned}$$

But an aggregation resource represents the average of all the resources it aggregates. Thus, the average cycle time for AR_1 is:

$$\overline{T_1^*} = \frac{T_1^*}{3} = \frac{468.8706}{3} = 156.2902$$

Hence, on average, a part processed by AR_1 spends a total of 156.2902 minutes waiting and being serviced.

Similarly, AR_2 represents all two server resources ($AR_2 = \{R_2, R_3\}$). The total cycle time of AR_2 is:

$$T_2^* = T_2 + T_3 = 178.187 + 247.071 = 425.259$$

and the average cycle time of AR_2 is:

$$\overline{T_2^*} = \frac{T_2^*}{2} = \frac{425.2580}{2} = 212.629 .$$

Since aggregation resource AR_4 represents a single resource ($AR_4 = \{R_4\}$), obviously both T_4^* and $\overline{T_4^*}$ are equal to T_4. Thus, $T_4^* = T_4 = 440.015$ and $\overline{T_4^*} = \frac{T_4^*}{1} = \frac{440.015}{1} = 440.015$.

For MPD's manufacturing flow line, the procedures of this section have provided a method for determining the cycle time of the original flow line resources. These values are then used to compute the average cycle time for each of the three aggregation

resources. The next section will use these estimates of average cycle time to determine the mean service time for each of the aggregate resources.

3.3 Service Mean of an Aggregation Resource

The previous section discussed a procedure for determining the average cycle time of an aggregation resource. Using this average cycle time, Section 3.3.1 determines the service mean that is necessary for create an aggregation resource with this given cycle time. This computed service mean will later be used by the aggregation methodology to weight the resource service time distributions of an aggregation resource. Section 3.3.2 continues the MPD example by finding the mean service time for the three aggregation resources.

3.3.1 Determining the Service Mean

The procedure for solving for the mean service time of an aggregation resource involves applying queueing formulas backwards. Most uses of queueing formula involve specifying the parameters (arrival rate, service mean, and capacity) of a resource or queueing system and computing the cycle or waiting time (such as was done in the previous section). Where this research differs is in that it seeks to specify the arrival rate, capacity, and cycle time of an aggregation resource with the objective of computing the mean service time. Hence, it is solving for the mean service time given the cycle time.

To understand this concept, consider a single server queueing system where service times are exponentially distributed. In this system, the M/M/1 queue performance can be summarized by the following formulas presented in Chapter 2:

$$\rho = \frac{\lambda}{\mu}$$

$$Po = 1 - \rho$$

$$Lq = \frac{\lambda\rho}{\mu - \lambda}$$

$$Wq = \frac{Lq}{\lambda}$$

$$W = \frac{1}{\mu - \lambda}$$

Thus, W (the cycle time) is equal to one divided by the service rate minus the service mean. Solving for μ, the service rate, results in:

$$\mu = \frac{1}{W} + \lambda .$$

That is, the service rate is equal to one divided by the cycle time plus the service rate. Solving for the mean service time, $1/\mu$, yields:

$$1/\mu = \frac{W}{1 + \lambda W} .$$

Since the value of λ is known from the definition of the flow line and W (cycle time) is computed (with the procedure of the previous section), the mean service time necessary to create an aggregation resource with cycle time W can be found.

Correspondingly, the procedure for the M/M/2 (two server) queueing system follows a similar development. As outlined in Chapter 2, the formula for computing the total weighting time (cycle time) with two servers is:

$$\rho = \frac{\lambda}{2\mu}$$

$$P_0 = \frac{1}{\left[\frac{(\lambda/\mu)^0}{0!} + \frac{(\lambda/\mu)^1}{1!}\right] + \left[\frac{(\lambda/\mu)^2}{2!(1-\rho)}\right]}$$

$$Lq = \frac{P_0(\lambda/\mu)^2\rho}{2!(1-\rho)^2}$$

$$W_q = \frac{L_q}{\lambda}$$

$$W = W_q + 1/\mu$$

Unfortunately, unlike the M/M/1 case, μ is not as easily computed. But by substitution, W can be expressed as:

$$W = \left[\frac{\frac{\lambda^3}{2\mu^3}}{2\lambda(1-\rho)^2\left[1+\frac{\lambda}{\mu}+\frac{(\lambda/\mu)^2}{2(1-\rho)}\right]}\right] + 1/\mu.$$

By simplification, this reduces to:

$$W = \frac{4\mu}{4\mu^2 - \lambda^2}.$$

Solving for μ requires using the quadratic formula and results in the following estimate of the mean service rate:

$$\mu = \frac{1 \pm \sqrt{1 + W^2\lambda^2}}{2W}.$$

Expressing this in terms of the mean service time (which must be greater than or equal to zero) results in the following estimate:

$$(1/\mu) = \frac{-2 + 2\sqrt{1 + W^2\lambda^2}}{W\lambda^2}.$$

Hence, the mean service time for a M/M/2 queueing system can be expressed in terms of it waiting time (cycle time) and arrival rate.

Unfortunately, the above results only apply to systems with exponential arrivals and exponential service times. By the assumption of independence, an arrival to an aggregation resource can be assumed to follow an exponential distribution. This combined with the fact that general service distribution times are allowed for means that a similar procedure must be followed but with the M/G/S queueing formula.

The previous section presented the M/G/S formula for computing the expected cycle time (T_j) of each resource. Summing each of these cycle times and dividing by the number of resources represented by an aggregation resource results in an estimates of $\overline{T_i^*}$, the average cycle time of an aggregation resources. This step of the aggregation methodology seeks is to estimate the service mean which creates an aggregation resource with cycle time, $\overline{T_i^*}$. The M/G/S queueing formula for computing this value of an aggregation resource is:

$$E\left[\overline{T_i^*}\right] = \frac{1+cv_{\delta_i}^2}{2\lambda(1-\rho_i^*)}\rho_i^* C_i^* + \delta_i^* \quad i=1,\ldots,O$$

where: $E\left[\overline{T_i^*}\right]$ Expected average cycle time of aggregate resource i (i = 1,...,O)

$cv_{\delta_i}^2$ Squared coefficient of variation of the unknown service time δ_i^* for aggregation resource i (i = 1,...,O)

λ Arrival rate of parts to the flow line

ρ_i^* Traffic intensity of aggregation resource i (i = 1,...,O): $\rho_i^* = \frac{\lambda\delta_i^*}{i}$

δ_i^* Mean service time of aggregate resource i (i = 1,...,O)

C_i^* $\frac{(\lambda\delta_i^*)^i}{i!(1-\rho_i^*)}P_{0_i}^* \quad i=1,\ldots,O$

$$P_{o_i}^* = P_{o_i}^* = \frac{1}{\left[\sum_{n=0}^{i-1}\frac{(\lambda\delta_i^*)^n}{n!} + \frac{(\lambda\delta_i^*)^i}{i!(1-\lambda\delta_i^*/i)}\right]} \quad i=1,\ldots,O.$$

As an example of this procedure, consider the case of aggregation resource AR_1. Since AR_1 represents all single server resources from the original flow line, determining

the mean serve time (δ_1^*) involves applying the M/G/1 queuing results backwards. Solving the M/G/1 formula for δ_1^* generates the following estimate:

$$\delta_1^* = \frac{\left(1+\lambda\overline{T_1^*}\right) \pm \sqrt{1+2\lambda^2-2\lambda\overline{T_1^*}+\lambda^2\overline{T_1^*}^2+2\lambda^2 cv_{\delta_1}^2}}{2\lambda}$$

With values for $\overline{T_1^*}$ (average aggregate resource cycle time) and λ (arrival rate) known, the only remaining unknown is the squared coefficient of variation ($cv_{\delta_1}^2$) of an aggregate resources service mean (δ_1^*). But since the mean is unknown (it is the quantity that this entire procedure is attempting to compute), a value of $cv_{\delta_1}^2$ must itself be estimated.

The procedure for estimating the squared coefficient of variation $cv_{\delta_i}^2$ for an aggregation resource is:

(1) Compute $cv_{m_j}^2 = \frac{s_j^2}{m_j^2} \quad j = 1,\ldots,N$

(2) Compute $cv_{\delta_i}^2 = \sum_{R_j \in AR_i} \left(\frac{T_j}{T_i^*}\right) cv_{m_j}^2 \quad \begin{matrix} i = 1,\ldots,O \\ j = 1,\ldots,N \end{matrix}$

That is, the squared coefficient of variation for an aggregation resource is a weighted average of the squared coefficient of variation of each of the service distributions aggregated by the aggregation resource. The weighting is a resource's cycle time. The procedure works by:

(1) Computing the squared coefficient of variation for each of the resources: $cv_{m_j}^2 = \frac{s_j^2}{m_j^2} \quad j = 1,\ldots,N$, where m_J^2 is the squared mean service time and s_j^2 is the variation associated with the resource service time distribution F_j.

(2) For each of the O aggregation resources, weight the squared coefficient of variation of a resource's service time by the proportion of that resource's cycle time (T_j) contribution of the aggregate resources total cycle time (T_i^*).

Applying the above procedure to estimate $cv_{\delta_i}^2$ allows the mean service rate to be determined for all aggregation resources.

This section illustrated several examples (M/M/1, M/M/2, and M/G/1) in how to work formulas backwards to find the necessary service mean for an aggregation resource with an arrival rate of λ and a cycle time of $\overline{T_i^*}$. The reader will note that this section does not present a table of formulas for larger capacity systems (*i.e.*, three or more servers). The reason for this omission is that the resulting formulas for the mean service time of these higher service capacity systems are extremely complex and each would take several pages to present. The approach of this research has been to demonstrate the technique for generating the formulas. As such, they can easily be implemented into a computer. Approaches for accomplishing this will be demonstrated in Chapter 4.

3.3.2 Mean Service Times for MPD Example

Applying the procedures Section 3.3.1 to the MPD manufacturing example results in estimates of the mean service time for each of the three aggregation resources. Consider aggregation resource AR_1. The previous section computed the average cycle time to be 156.2902. Before computing the mean service time needed to create a resource with cycle time of 156.2902, the squared coefficient of variation of the service time for AR_1 must be estimated. This involves weighting the squared coefficient of variation of each resource's service time by the percentage of that resource's cycle time to the overall total cycle time of the aggregation resource. For AR_1, the estimate of $cv^2_{\delta_1}$ is:

$$cv^2_{\delta_1} = \left[\left(\frac{240.208}{468.8706}\right).00130208\right] + \left[\left(\frac{63.7106}{468.8706}\right).0163786\right] + \left[\left(\frac{164.952}{468.8706}\right).00411523\right]$$
$$= .00434038$$

Using this value, the mean service rate of AR_1 (δ_1^*) can be found. Solving results in δ_1^* being equal to 70.6877.

Results of computing the mean service time for the other aggregation resources are summarized in Table 3.8. The interesting case to note is AR_4. Since this aggregation resource represents only a single resource, the estimate of the squared coefficient of variation of the aggregation service mean is the squared coefficient of variation of the single resource service mean.

Table 3.8. Estimates of the mean service time and squared coefficient of variation for each of the three aggregation resources.

	AR_1	**AR_2**	**AR_4**
Cycle Time (T_i^*) =	468.871	425.259	440.015
Ave. Cycle Time ($\overline{T_i^*}$)=	156.29	212.629	440.015
Est. SQ. C of V ($cv^2_{\delta_i}$) =	.00434038	.0070735	.003125
Est. Service Mean (δ_i^*) =	70.6877	141.3730	320.000

The techniques of this section have provided a means for computing the service mean necessary required by an aggregation resource with a specified average cycle time. This service mean will next be used to develop a procedure for weighting the original resource service means.

3.4 Resource Weighting Procedure

The previous sections of this chapter summarized procedures for computing the total waiting time represented by an aggregation resource and summarized methods for computing the average service mean of an aggregate resource. Section 3.4.1 continues the aggregation methodology by developing a procedure for weighting the service time means of the resources aggregated by an aggregation resource. Section 3.4.2 illustrated this procedure on the MPD manufacturing example.

3.4.1 Determining Resource Weights

The weights developed in this section represent the percentage contribution of each resource service mean towards an aggregation resource service mean. These weights, must satisfy two conditions: (1) the sum of all the resource weights multiplied by the original resource mean service time is equal to the average service time of the aggregation resource (δ_i^*) that was computed by the procedures of the previous section and, (2) the sum of the weights is equal to one. More formally, these two conditions are:

$$(1)\ \sum_{R_j \in AR_i} w_j^* m_j = \delta_i^* \quad \begin{matrix} i = 1,\ldots,O \\ j = 1,\ldots,N \end{matrix} \quad \text{and} \quad (2)\ \sum_{R_j \in AR_i} w_j^* = 1 \quad \begin{matrix} i = 1,\ldots,O \\ j = 1,\ldots,N \end{matrix}$$

This convex relationship thus determines the proportional weight that each resource service mean contributes towards the average service time of the aggregation resource. In the next section, these weights will be used in combination with the original resource service distributions (F_i) to develop a procedure for estimating the service time distributions of the aggregation resources (F_i^*).

The easiest case to determine distribution weights for is one in which an aggregate resource represents a single resource. In such a case, since the aggregation resource represents a since resource, the aggregate service mean (δ_i^*) is merely the resource service mean, m_j, where $R_j \in AR_i$. Thus, the distribution weight for the resource service

mean, w_j^* of resource R_j is 1.0. Clearly this satisfies the two weighting conditions: $w_j * m_j = \delta_i^*$ and $w_j^* = 1.0$.

The case of determine the distribution weights for when two resources is aggregated together is similarly easy. Here, the aggregation resource (*e.g.*, AR_2) represents the aggregation of two resources (*e.g.*, R_2 and R_5). Our objective in determining the weights is to decide how to weigh the two individual service resource means (m_2 and m_5) such that they equal the aggregate service mean (δ_2^*). Applying the two weighting conditions results in the following equations:

$$(1)\ (w_2^* \times m_2) + (w_5^* \times m_5) = \delta_2^*,\ \text{and}\ (2)\ w_2^* + w_5^* = 1.$$

Since the values of m_2, m_5, and δ_2^* are known, the task of solving for w_2^* and w_5^* simply involves applying standard algebraic procedures for solving two equations with two unknowns.

By following similar logic, considers what occurs when an aggregation resource consists of three resources. To determine the distribution weights reduces to having to solving two equations and with three unknowns. For example:

$$(1)\ (w_1^* \times m_1) + (w_3^* \times m_3) + (w_5^* \times m_5) = \delta_4^*,\ \text{and}\ (2)\ w_1^* + w_3^* + w_5^* = 1.$$

In this instance, the solution can only be reduced to a set of relationships among the variables. Determining a more specific solution requires much trial and error. Consider what happens when an aggregate resource represents (say) 20 resources. Here, the current solution technique involves solving 2 equations with 20 unknowns (the weight for each of the 20 resource service means). Quite a difficult, if not impossible task!

The technique to determine the service time weighting for the case when three or more resources are represented by an aggregation resource must be expanded. The expanded procedure combines the techniques of the previous sections (determining total cycle time and deriving the average aggregate resource service mean) with a recursive

algorithm to reduce (by aggregating) the P_i resources of an aggregation resource to only two resources. In essence, it aggregates within the aggregation resource to reduce the resources represented by the aggregation resource to only two, much as how the original system of N resources were combined. As demonstrated, determining the distribution weights for an aggregation resource representing two resource is easily derived by solving a set of two equations with two unknowns.

To illustrate this recursive technique, consider the case (Figure 3.7a) of aggregation resource (AR_2) which represents the aggregation of 5 resources (R_1, R_3, R_4, R_6, and R_7). Each of these resources is characterized by its service mean (m_1, m_3, m_4, m_6, and m_7). Without an expanded procedure, determining the distribution weights requires solving:

$$(w_1^* \times m_1) + (w_3^* \times m_3) + (w_4^* \times m_4) + (w_6^* \times m_6) + (w_7^* \times m_7) = \delta_2^*, \text{ and} \qquad (3.1)$$

$$w_1^* + w_3^* + w_4^* + w_6^* + w_7^* = 1.$$

As discussed prior, solving these two equations of five unknowns is a difficult task.

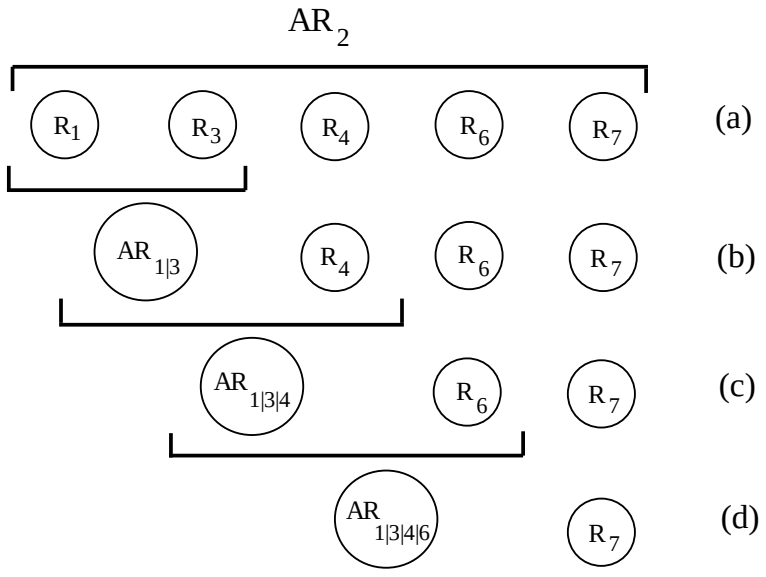


Figure 3.7. Recursive procedure to determine the distribution weight for an aggregation resource consisting of three or more resources.

The first step of the recursive algorithm is to aggregate two resources (*e.g.*, R_1 and R_3). This is done by determining the total cycle time of the two resources (T_1 and T_3), and divide this by two, to find the average cycle time of the "new" aggregate resource. That is, the average cycle time of aggregate resource, $A_{1|2}$ (an aggregate resource within an aggregation block) is $\overline{T^*_{1|3}}$, where $\overline{T^*_{1|3}}$ = (T_1+T_3)/2. Next, using the techniques of the previous section, compute the mean service time ($\delta^*_{1|3}$) for a resource with average cycle time $\overline{T^*_{1|3}}$.

This aggregation (Figure 3.7b) reduces the number of distinct resources represented by the aggregation resource by one (since two were aggregated together). Thus, determining the weights reduces to solving:

$$(w^*_{1|3} \times \delta^*_{1|3}) + (w^*_4 \times m_4) + (w^*_6 \times m_6) + (w^*_7 \times m_7) = \delta^*_2, \text{ and} \qquad (3.2)$$
$$w^*_{1|3} + w^*_4 + w^*_6 + w^*_7 = 1$$

where $w_{1|3}{}^*$ is the weight and $\delta^*_{1|3}$ is the average service time computed for the aggregate resource resulting in aggregating R_1 and R_3. This aggregation process should continue until only two resources are represented by AR_2, where one is an original system resource and the other is an aggregate resource itself.

Continuing this example, by aggregating an addition resources (R_4), determining the distribution weights reduces to solving (Figure 3.7c):

$$(w^*_{1|3|4} \times \delta^*_{1|3|4}) + (w^*_6 \times m_6) + (w^*_7 \times m_7) = \delta^*_2, \text{ and} \qquad (3.3)$$
$$w^*_{1|3|4} + w^*_6 + w^*_7 = 1.$$

where $w_{1|3|4}{}^*$ is the weight and $\delta^*_{1|3|4}$ is the average service time computed for the aggregate resource resulting from aggregating R_1, R_3, and R_4.

By continuing to incrementally aggregate an additional resource (R_6), there are now only two resources represented by the AR_2 (one of which is an aggregation resource itself). This system generates the following equations (Figure 3.7-d):

$$(w^*_{1|3|4|6} \times \delta^*_{1|3|4|6}) + (w^*_7 \times m_7) = \delta^*_2, \qquad (3.4)$$

and $w^*_{1|3|4|6} + w^*_7 = 1$.

At this point, since the values for m_7 is known and the values of $\delta^*_{1|3|4|6}$ and δ^*_2 have been previously computed, the above equations (3.4) can easily be solved using the standard algebraic techniques for solving two equations with two unknowns. Doing so results in distribution weights which represent the proportionally weight of each service time distribution to generate an aggregate service mean of δ^*_2 The value computed for w_7^* is the distribution (or percentage) weight that m_7 adds towards an aggregate resource service mean of δ^*_2. The value for $w_{1|3|4|6}^*$ is the percentage weight of all the other (aggregated) resources of the aggregation resource.

The reason this approach has been termed recursive is that now that the problem has been reduced to a point in which it can be solved, the procedure works incrementally backwards using this and subsequent solutions to solve the previous level of resource aggregation. This process continues until all original resources represented by the aggregation resource have distribution weights.

Thus, since a value for w_7^* is known from solving (3.4), the equations (3.3) reduce to a set of two equations with two unknowns. Solving these generates a value for w_6^*. With this value, the equations in (3.2) reduces to two equation and two unknowns and the value of w_3^* can be found. With this value, equations (3.1) can be solved to provide a value of w_1^*.

Applying this recursive algorithm has found all the distribution weight values for the case when the aggregation resource represents three or more resources. The general approach to determine the distribution weighting for any aggregate resource (AR_i) is presented in Table 3.9.

Table 3.9. Procedure for determining the resource weightings of an aggregation resource.

PROCEDURE Weight (resources aggregated by AR_i (*i.e.*, P_i), aggregation resource mean, total probability)

(* AR_i represents one resource *)
IF (P_i = 1) THEN
(* Resource weight is equal to 1*)
w_j^* = 1, where $R_j \in AR_i$
END IF

(* AR_i represents two resource with unknown weights *)
IF (P_i = 2) THEN
(* Solve the two equations of two unknowns for their weights *)

$$(w_j^* \times m_j) + (w_k^* \times m_k) = \delta_i^*$$

$$w_j^* + w_k^* = 1, \quad \text{where } R_j \text{ and } R_k \in AR_i$$

END IF

(* AR_i represents three or more resources, need to aggregate *)
IF ($P_i \geq 3$) THEN
Aggregate 2 resources within AR_i
Determine the average service time of the new resource
Call Weight (resources aggregated by AR-1 (*i.e.*, P_i - 1), AR mean, total probability)
Adjust aggregate resource mean and total probability
Call Weight (resources aggregated by AR, adjusted AR mean, adjusted total probability)
END IF

END PROCEDURE

The interested reader is referenced to Appendix B where the recursive algorithm is implemented in the Mathematica modules DistWeight1 and DistWeight2. Discussion of these modules and their implementation will be discussed in Chapter 4.

3.4.2 Resource Weights for MPD Example

This fourth step of the aggregation methodology uses the mean service time of a resource to determine its contribution towards the aggregate service time mean. For the MPD manufacturing system, weights must be determined for each of the six resource service time means. The easiest case is aggregation resource AR_4. In this instance, AR_4 has a service mean of δ_4^* = 320 and represents a single resource with a mean service time of 320. Due to the aggregation resource representing a single resource, the weight of w_4^* is:

$$320 w_4^* = 320$$
$$w_4^* = 1$$

Solving yields $w_4^* = 1$. Thus, the weight of m_4 towards δ_4^* is 1.0 (one hundred percent).

For aggregation resource AR_2, δ_2^* was previously computed to be 141.373. Since AR_2 represents two resources, determining the weights requires solving:

$$130 w_2^* + 150 w_3^* = 141.373$$
$$w_2^* + w_3^* = 1$$

Solving the set of two equations with two unknowns yields w_2^* = .431361 and w_3^* = .568639. This means that m_2 (130) contributes 43.1361% towards the aggregate service mean of 141.373 and m_3 (150) contributes the remaining 56.8639%.

Aggregation resource AR_1 represents three resources. As such, determining the weights involves applying the recursive procedure. It was previously computed that δ_1^* was equal to 70.6876. Determining the weights requires solving:

$$80 w_1^* + 45 w_5^* + 72 w_6^* = 70.6877$$
$$w_1^* + w_5^* + w_6^* = 1$$

As discussed, solving these two equations and three unknowns requires using a recursive algorithm. The first task is to aggregate two of the resources within the aggregation resource. Thus, aggregating (applying the techniques of Sections 3.2 and 3.3) R_1 and R_5

yields a new aggregation resource: $AR_{1|5} = \{R_1, R_5\}$. The total cycle time of this resource is:

$$T_{1|5}^* = T_1 + T_5 = 240.208+63.7106 = 303.9186$$

The average cycle time of $AR_{1|5}$ is:

$$\overline{T_{1|5}^*} = \frac{T_{1|5}}{2} = \frac{303.9186}{2} = 151.95930$$

To determine the mean service time needed to generate an average cycle time of 151.9593 requires estimating the squared coefficient of variation:

$$cv_{1|5}^{2^*} = \left[\left(\frac{240.208}{303.9186}\right).00130208\right] + \left[\left(\frac{63.7106}{303.9186}\right).0163786\right]$$
$$= .004463$$

Using this value, the mean service time is computed to be $\delta_{1|5}^*$ = 69.9882.

Now that R_1 and R_5 have been aggregated, the aggregation resource reduces to $AR_1 = \{AR_{1|5}, R_6\}$. Thus, the aggregation resource represents two resource, $AR_{1|5}$ which has a service mean of 69.9882 and R_6 with a service mean of 76. With only two resources represented, the weights can be determined:

$$69.9882w_{1|5}^* + 72w_6^* = 70.6877$$
$$w_{1|5}^* + w_6^* = 1$$

Solving yields $w_{1|5}^* = .652321$ and $w_6^* = .347679$. Thus, the contribution of m_6 towards the aggregation resource service time is 34.7679%, while the other (currently aggregated) resources contributes 65.23211%.

The next step, now that a value of w_6^* is known, is to go to the to the previous level of aggregation and plug this value into the equations:

$$80w_1^* + 45w_5^* + 72(.347679) = 70.6877$$
$$w_1^* + w_5^* + .347679 = 1$$

These equations reduce to:

$$80w_1^* + 45w_5^* = 45.654812$$
$$w_1^* + w_5^* = .652321$$

Solving yields the values: $w_1^* = .465724$ and $w_5^* = .186597$. Note that the sum of w_1^*, w_5^*, and w_6^* is 1.00.

These weights (summarized in Table 3.10) will next be used to develop the aggregate simulation model of the flow line system. Each weight will represent the weight of the resource service time distribution in estimating the aggregation resource service time distribution.

Table 3.10. Summary of distribution weights for MPD example.

Aggregate Resource #1	**Distribution Weight**
Resource #1	.465724
Resource #5	.186597
Resource #6	.347679

Aggregation Resource #2	**Distribution Weight**
Resource #2	.431361
Resource #3	.568639

Aggregation Resource #4	Distribution Weight
Resource #4	1.0

3.5 Aggregate Simulation Model

In the previous sections of this chapter, the flow line system used for this research has been defined, procedures to compute weighting times for a resource have been summarized, techniques for computing the service time of an aggregate resource have been presented, and a method to weight the resource service time means to estimate an aggregation resource service mean has been developed.

The final task is to develop an aggregate simulation model for estimating part cycle time. Section 3.5.1 summarizes the statistics that must be collected and specifies how the aggregate simulation should be defined. Section 3.5.2 concludes the MPD example by developing the specifications of the aggregate simulation model.

3.5.1 Developing the Aggregate Simulation Model

In the aggregate simulation model, each of the aggregation resources is explicitly modeled to represent the average of all the resources it has aggregated. Previously, Figure 3.5 described this system. Given the assumption of aggregate resource independence, the technique needed to estimate the cycle time of a part involves modeling each of the aggregate resources in the aggregate simulation model, where each aggregation resource is modeled as a standard “queue-seize-delay-release” simulation relationship.

Since aggregation capabilities are not included in most simulation languages, the computation of the average cycle time for a part is a challenging task. Assume that by running the aggregate simulation model a total of r parts “flow” through the model. If so, Table 3.11 summarizes the four types of data that must be collected.

Table 3.11. Statistics needed to be collected by aggregate simulation model.

$\overline{Y_{ij}}$ = Average Cycle Time of Part # j on AR_i $\quad i = 1,\ldots,O$, $j = 1\ to\ r$
Y_{ij} = True Cycle Time of Part # j on AR_i $\quad i = 1,\ldots,O$, $j = 1\ to\ r$
Z_j = Total Cycle Time of Part # j $\quad j = 1\ to\ r$
$\overline{Z}$ = Average Cycle Time for all r Parts

Since an aggregate resource is an average of all the original resources (R_i), the cycle time of a part through an aggregation resource (AR_i) is really an average of the true aggregate resource processing time ($\overline{Y_{ij}}$). The true processing time of an aggregate resource is $Y_{ij} = P_i * \overline{Y_{ij}}$, where P_i is the number of resources aggregated in AR_i. Correspondingly, the total cycle time of part #j (j = 1,...,r) is equal to the sum of the processing and waiting time that part #j spends at all the O aggregate resources. Hence,

$$Z_j = \sum_{i=1}^{O} Y_{ij} \quad j = 1,\ldots,r$$

where Z_j is the total cycle time of a part.

The final statistic to be collected or computed is the average cycle time of all r parts which "flow" through the simulation model. That is,

$$\overline{Z} = \frac{\sum_{i=1}^{r} Z_j}{r}$$

By knowing which statistics are to be collected, the next step is to determine how to estimate the service distribution (F_i^*) of an aggregation resource. The solution is to use

the weights, w_i^* (i = 1 to N), developed in the previous section. These distribution weights are the key to creating an aggregate simulation model of a manufacturing system.

Consider any aggregate resource, AR_i. This resource is characterized by F_i^*, its service time distribution, which represents the P_i system resource service distributions (F_i) that have been aggregated to form AR_i. Unfortunately, since the individual service distributions (F_i) can be any general distribution, developing a combined or joint distribution of the aggregate service time density function (f_i^*) may be neither feasible, efficient, nor possible. An alternative is to represent this unspecified service time distribution not as a mathematical function, but rather as a relationship that random numbers can be sampled from.

What this research requires is a method for generating a random variable from F_i^* (the aggregation resource service time distribution) during the execution of the aggregate simulation model. As discussed above, this distribution represents all the service time distributions an aggregation resource represents. Unfortunately, deriving this joint distribution is not feasible for most practical problems due to the time, complexity, and feasibility involved. Thus, the aggregate service time distribution is unknown. But the individual service time distributions (F_i) composing F_i^* (whose mathematical representation is unspecified) plus the weights (w_i^*) specifying the importance (contribution) of each resource service time mean (m_i) toward an aggregation resource are known. With only this information, an effective solution technique is to use a procedure known as the *composition* or *mixture method* for generating random variables. This procedure was introduced in Chapter 2. Thus, the aggregate resource service time distribution is never specified, but rather, values from it will be sampled during the execution of the aggregate simulation model.

Pritsker (1986) summarizes this technique assumes that "the density function can be written as a weighted sum of component distribution functions with the sum of the weights totaling one." Kronmal and Peterson (1979) continue and explain that some continuous distribution are efficiently generated by representing them as mixtures of several other (continuous) distributions that are easy to generate. For this research, this is exactly the case, in that the aggregate service distribution (F_i^*) must be estimated (a very difficult task), but is defined in terms of the service time distributions (F_i) (which are easier to estimate) making up the aggregation resource. In addition, the original service time distributions are related to the aggregate service time distribution by the distribution weights (w_i^*) derived in the previous section, which were defined to sum to one.

In terms of this research, each aggregation block (AR_i) density function (f_i^*) can be defined with the following functions and weighting relationship among the distributions:

1. the number $P_i \geq 1$ of resources aggregated in AR_i
2. the distribution weights $w_1^*, w_2^*, ..., w_N^*$. Subject to:

$$w_j^* \geq 0, \quad R_j \in AR_i \quad j = 1, \ldots, N$$

$$\sum_{R_j \in AR_i} w_j^* = 1 \quad j = 1, \ldots, N$$

3. the elements (density functions) $f_1, f_2, ..., f_n$ aggregated by AR_i, subject to the constraint that the mixture of the density functions f_i, i = 1, 2, ..., P_i satisfies:

$$f_i^*(x) = \sum_{j=1}^{P_i} w_j^* f_j(x) \quad i = 1, \ldots, O$$

The general composition algorithm for sampling the service distribution of an aggregate resource (AR_i) is as follows:

1. Generate a positive random integer J such that

$$P(J = j) = w_j^* \quad (j=1,2,...)$$

2. Return the random variable, X, with a sample from density function f_J.

Thus, to generate a random sample for an aggregation resource service time (f_i^*), one first uses an inverse transformation method (dependent upon the simulation language) to select one of the component density functions (f_J). A sample, X, is drawn from this distribution (F_J) and it becomes the sample for the aggregate distribution (F_i^*). By repeated sampling, each of the component distributions are selected in accordance with their weights and, hence, the samples are generated in accordance with the aggregate service time distribution (Law and Kelton, 1991).

3.5.2 Aggregation Simulation for MPD Example

For the MPD manufacturing flow line, three aggregation resource (AR_1, AR_2, and AR_4) must be represented in the aggregate simulation model.

The topology of the aggregate simulation model was depicted in Figure 3.5. The time between parts arriving follows an exponential distribution with a mean of 100 minutes. Upon arrival, a part is sent to each of the aggregate resources, where it waits for service and is eventually serviced. Upon completion, the part leaves the system. Throughout this process, the statistics identified in Table 3.9 must be collected to estimate the average cycle time of a part.

The final component of the aggregate simulation model is the procedure for estimating the service time distribution of an aggregation resource. The presentation of this section introduced a procedure for solving this. It showed that the service time distribution of an aggregation resource can be estimated by the service time distributions of the resources it aggregates in combination with the distribution weights computed in the previous section.

For instance, AR_4 represents a single resource, R_4, with service distribution F_4 = Normal (320, 25) and weight $w_4^* = 1.0$. The service distribution of AR_4 is:

$$F_4^*(I) = \{Normal(320,25) \quad \text{for all I}$$

where I is a uniform random variable with minimum value 0 and maximum 1 that is generated when a sample is sought (*i.e.*, a service time must be assigned) from F_4^*.

AR_2 represents two resources, R_2 and R_3 with service distributions F_2 = Normal (130, 15) and F_3 = Triangular (120, 150, 180) and weights w_2^* = .431361 and w_3^* = .56839. The service distribution of AR_2 is:

$$F_2^*(I) = \begin{cases} Normal(130,15) & 0 \le I < .431361 \\ Triangular(120,150,180) & .431361 \le I \le 1 \end{cases}$$

where I is a Uniform (0,1) random variable that is generated when a sample from F_2^* is required.

Resources R_1, R_5, and R_6 are represented by AR_1. These resources are characterized by their service distributions, F_1 = Uniform (75,85), F_5 = Triangular (32, 43, 60), and F6 = Uniform (64, 80), and their distribution weights, w_1^* = .465724, w_5^* = .186597, and w_6^* = .347675. The service distribution of AR_1 is:

$$F_1^*(I) = \begin{cases} Uniform(75,85) & 0 \le I < .465724 \\ Triangular(32,43,60) & .465724 \le I < .652321 \\ Uniform(64,80) & .652321 \le I \le 1 \end{cases}$$

where I is a Uniform(0,1) random number that is generated when a sample from F_1^* is needed.

Using this specification, the aggregate simulation model can be developed to estimate the cycle time of the MPD manufacturing system. Appendix A presents a listing of the SLAM simulation model of the full (non-aggregated) MPD flow line system along with its aggregate equivalent.

3.6 Chapter Summary

The objective of this chapter was to present the aggregation methodology for creating an aggregate simulation model for estimating part cycle time. The specific steps of the methodology are:

(1) Describe the manufacturing system. From the description of the "as is" system, an aggregate description can be developed by aggregated all same capacity resources into aggregation resources.

(2) Estimate the average cycle time for each aggregation resource. This estimate is the average of all the cycle times of the resources aggregated by an aggregation resource.

(3) Find the mean service rate of each aggregation resource. This is found by applying queueing formulas backwards, in that the mean service time is for given the arrival rate and cycle time.

(4) Determine the distribution weights. Using the mean service time of an aggregation resource, the percentage contribution of each resource's mean toward this value is determined using a recursive algorithm.

(5) Specify the aggregate simulation model. With the distribution weights, use composite sampling to specify an aggregate simulation model which replicates the original system.

It is hypothesized that applying this methodology results in an aggregate simulation model which accurately estimates the average part cycle time when compared to the full model. An evaluation of this procedure is presented in the next chapter.

CHAPTER 4

APPLICATION OF AGGREGATION METHODOLOGY

The objective of the aggregation methodology is to generate the specifications necessary for creating an aggregate simulation model for approximating the average cycle time of a part through the flow line. Several methods will be used to demonstrate the effectiveness of applying the methodology. Section 4.1 divides the analysis of the aggregation methodology into three specific types of flow lines: a flow line with all exponential servers, a flow line with only single capacity servers, and a flow line with multiple servers of any service distribution. Section 4.2 discusses the development of a computer program which implements the aggregation methodology for finding the specifications of an aggregate simulation model. Section 4.3 illustrates using the methodology for each of the three types of flow line systems.

4.1 Impact of the Aggregation Methodology

The key component of the aggregation methodology is the procedure used for estimating the cycle time for each of the N resources. These cycle time values are summed and averaged to compute the average cycle time for each of the O aggregation resources. The average aggregate resource cycle time are used to determine the mean service time needed for each of the O aggregation resources. Next, distribution weights are determined from computing the aggregation resource service means in terms of the resource service means. With these weights, running the aggregate simulation model attempts to replicate each of the average aggregate resource's cycle times. But since this value is based on the individual resource cycle time, the entire aggregation procedure is dependent upon these original estimates.

In Chapter 3, based on the work of Kleinrock (1976), the cycle time for each resource was computed using a M/G/S queueing formula. Doing so allowed this research

to assume that the arrival distribution to any resource in the flow line is Poisson, and thus can be described by an exponential distribution. In actuality this is not true. To understand, image a flow line system. By definition, parts arrive to R_1 (the first resource of the flow line) in a Poisson fashion, and can be modeled by an exponential distribution. Thus, the mean and standard deviation of the arrival process are equal. That is, the coefficient of variation of the arrival process, cv_a, is equal to one. Computing the cycle time for R_1 is appropriately computed by the M/G/S queueing formula. Once a part finishes being processed at R_1, it goes to R_2. Obviously, during steady state, the mean time between arrivals to R_2 is the same as the time between arrivals to R_1, but if the service time of R_1 is not exponential, then the standard deviation (or variation) of the arrival time to R_2 will not be equal to the mean. In fact, it will later be shown in Section 4.1.3 that the coefficient of variation of the service time is less than or equal to one. Thus, the arrival process to R_2 is probably not Poisson. The remainder of the section seeks to address this dilemma.

The aggregation methodology's success will be judges on its effectiveness for three types of systems. Section 4.1.1 discusses the application of the aggregation methodology for the case when resource service times are all exponentially distributed. Section 4.1.2 presents an extension to the aggregation methodology for the case when a flow line consists of only single capacity resources. Section 4.1.3 determines that applying the aggregation methodology to a flow line system with the assumption of independence among resource results in an upper bound of the expected cycle time.

4.1.1 Flow Line with Exponential Service Times

In the special case when all servers (of any service capacity) in the flow line have exponential service time distributions, then the aggregation methodology will create an aggregate simulation model which approximates the average cycle time quite well. The

reason for this is that with all service time distributions being exponential, each resource cycle time (the key component of the aggregation methodology) will be computed with the M/G/S queueing formula. This formulas is exact for estimating the steady state performance of a M/M/S resource (queueing system).

Burke (1956) showed that the output process of a M/M/S queue is itself Poisson with a mean equal to its arrival mean. Thus, in a flow line system with all exponential service times, the arrival process to any resource is Poisson. Hence, the problem of changing arrival time variability is not relevant for this special case. In fact, because the arrival process to each resource is the same, the assumption of independence is in fact true.

Even with all exponential service times, the aggregation methodology works exactly as before. First the system is defined and then the resource are aggregated into their appropriate aggregation resources. Next, the cycle time for each resource is computed. These will be exact estimates of the steady state performance. These cycle time values will be summed and an average aggregation resource cycle time will be computed for each of the aggregation resources. The service mean necessary to create each of the aggregation resource's average cycle times will be computed. To accomplish this requires an estimate of the squared coefficient of variation for each of the aggregation resources. As outlined in Section 3.3, this is computed as a weighted average of the squared coefficient of variations of the resource times. But since the service times are all exponential and have a coefficient of variation of one, then a weighted estimate of them will also be one. This seems logical in that if an aggregation resource represents all exponential resources, then the aggregation resource service time distribution will also be exponential (*i.e.*, have a coefficient of variation equal to one).

The M/G/S queue results are applied backwards to find the mean service time resulting in an exact estimate of the steady state mean service time necessary to create an

M/M/S aggregation resource with the given average cycle time. This mean value will be used to determine the distribution weights of each of the resource service time distributions. Finally, developing an aggregate simulation model with the distribution weights attempts to replicate each of the average aggregate resource cycle times. Since these cycle time values are exact at steady state, running the resulting aggregate simulation model should provide an accurate estimate of the cycle time.

4.1.2 Flow Line with Single Service Capacity Resources

If a flow line consists of only single capacity servers for all resources, applying the extension of this section will create an aggregate simulation model which should approximate the original flow line system. Obviously, the aggregate equivalent of a flow line consist of only single server capacity resources is a single aggregation resource, namely AR_1.

The extension necessary for accomplishing this is to use a more detailed procedure for computing the cycle time of an aggregation resource. As such, the assumption of independence of Chapter 3 is not required, for a resource's order now has a vital role in determining an estimate of each resource's cycle time.

This research assumes that parts arrive to the first resource, R_1, by a Poisson process. Thus, the cycle time for a part at R_1 can be estimated by the Pollaczek-Khinchine formula (discussed in Section 2.3.2) for an M/G/1 queue (Kleinrock, 1976):

$$E[T_1] = \frac{\lambda\left(m_1^2 + \sigma_{m_1}^2\right)}{2(1-\rho)} + m_1$$

where: $E[T_1]$ Expected cycle time for the first resource (R_1)

λ Arrival rate to R_1

m_1 Average service time of R_1.

σ_m^2 Service time variation for R_1

ρ Traffic intensity of R_1

Burke (1955) showed that the output of a M/M/S queue is Poisson. Thus, if the service distribution of R_1 is exponential, its output process (arrival process to R_2) will also be Poisson with the same parameter values. But since general service time distributions are allowed, this condition only aides the special case when exponential service time distributions occur or when independence is assumed.

Without these conditions, a G/G/1 (general arrival and general service) queueing formula must be used for estimating the cycle time for subsequent resources in the flow line. As presented in Section 2.3.3, Kumura (1991) proposes the following approximation:

λ Mean arrival rate

cv_a^2 Coefficient of variation of the arrival time

μ Mean service rate

cv_t^2 Coefficient of variation of the service time

ρ Traffic intensity: $\rho = \dfrac{\lambda}{\mu}$

g $$g(\rho, cv_a^2, cv_t^2) = \begin{cases} Exp\left[-\dfrac{2(1-\rho)}{3\rho} \dfrac{(1-cv_a^2)^2}{cv_a^2 + cv_t^2} \right], & cv_a^2 \le 1 \\ 1, & cv_a^2 > 1 \end{cases}$$

$E_{M/M/1}[Wq]$ Expected queue waiting time for a M/M/1 queue:

$$E_{M/M/1}\left[W_q\right] = \frac{\rho}{\mu - \lambda}$$

$E_{G/G/1}[Wq]$ Expected queue waiting time for a G/G/1 queue:

$$E_{G/G/1}\left[W_q\right] \approx \frac{cv_a^2 + cv_t^2}{2} g\left\{E_{M/M/1}\left[W_q\right]\right\}$$

Rewriting this formula in terms of the flow line methodology results in the following resource cycle time estimator:

$$E[T_i] = \left\lceil \frac{cv_{a_i}^2 + cv_{m_i}^2}{2} g\left\{ \frac{\rho_i}{1/m_i - \lambda} \right\} \right\rceil + m_i \quad i = 2, \ldots, N$$

This formula specifically requires the rate of arrival to a resource and the squared coefficient of variation of the arrival process. To compute the coefficient of variation of the arrival process requires knowing the variation of the arrival process.

To determine this quantity, it is necessary to explore the output process of a single server queue. First, some notation must be defined (Medhi, 1991):

Let: $t_n \equiv$ instant of arrival of the nth part to the resource

$u_n \equiv$ interarrival time between the nth and (n+1)th part = $t_{n+1} - t_1$

$v_n \equiv$ service time of the nth part

$X_n \equiv v_n - u_n$

$W_n \equiv$ waiting time in the queue of the nth part

$D_n \equiv$ instant of departure of the nth part

$I_{n-1} \equiv$ idle time (if any) proceeding the nth arrival.

With these definitions, the interdeparture interval, τ_n, between the nth and (n+1) departure of the queue (resource) is given by (Marshall, 1968):

$$\begin{aligned} \tau_n &= D_{n+1} - D_n \\ &= t_{n+1} + w_{n+1} + v_{n+1} - (t_n + w_n + v_n) \\ &= (t_{n+1} - t_n) + w_{n+1} - w_n + v_{n+1} - v_n \end{aligned}$$

But, in steady state, $E(W_{n+1}) = E(W)$, so the expected interdeparture interval is given by:

$$\begin{aligned} E(\tau_n) &= E(t_{n+1} - t_n) \\ &= E(u_n) \\ &= \frac{1}{\lambda} \end{aligned}$$

Therefore, in steady state, the time between the arrival of parts to subsequent resource in the flow line is the same as the arrival process. Hence, the mean time between arrivals does not change and remains constant throughout the flow line. Unfortunately, because

the resource service time distributions are not required to be as variable as the exponential service time distribution, the variability of the arrival times does change (in fact, it usually decreases).

To estimate the change in the arrival time variability for subsequent resources of the flow line. Marshall (1968) shows that the variance of the interdeparture interval (output process) is:

$$Var(\tau_n) = \sigma_v^2 - \frac{(1-\rho)^2}{\lambda^2} + \left\lceil \frac{1-\rho}{\lambda} \right\rceil \left\lceil \frac{v_h^{(2)}}{v_h} \right\rceil$$

The proof is as follows:

$$\tau_n = v_{n+1} + I_n$$

where v_{n+1} and I_n are independent. Hence,

$$Var(\tau_n) = Var(v_{n+1}) + Var(I_n).$$

Again,

$$\begin{aligned} W_{n+1} - I_n &= W_n + X_n = W_n + u_n - v_n \\ Var(W_{n+1} - I_n) &= Var(W_n) + Var(u_n) + Var(v_n) \\ &= \sigma_W^2 + \sigma_u^2 + \sigma_v^2 \end{aligned}$$

But,

$$Var(W_{n+1}) = Var(W_{n+1}) + Var(I_n) - 2Cov(W_{n+1}I_n).$$

Now $W_{n+1}I_n = 0$, and hence

$$\begin{aligned} Cov(W_{n+1}I_n) &= E(W_{n+1}I_n) - E(W_{n+1})E(I_n) \\ &= -E(W)\left\lceil \frac{1}{\lambda} - \frac{1}{\mu} \right\rceil \end{aligned}$$

Substituting, we get

$$\sigma_W^2 + \sigma_u^2 + \sigma_v^2 = \sigma_W^2 + Var(I_n) + 2E(W)\left(\frac{1}{\lambda} - \frac{1}{\mu}\right)$$

so that

$$Var(I_n) = \sigma_u^2 + \sigma_v^2 - 2\left(\frac{1}{\lambda} - \frac{1}{\mu}\right)E(W)$$

and combining, we get

$$Var(\tau_n) = \sigma_u^2 + 2\sigma_v^2 - \frac{2}{\lambda}(1-\rho)E(W) \qquad (4.1)$$

Using the definition of E(W), we get:

$$Var(\tau_n) = \sigma_v^2 - \frac{(1-\rho)^2}{\lambda^2} + \left\lceil \frac{1-\rho}{\lambda} \right\rceil \left\lceil \frac{v_h^{(2)}}{v_h} \right\rceil$$

But estimating the values of v_h and $v_h^{(2)}$, the first and second moments of the idle period I is extremely difficult. A solution is to use equation (4.1), which is in terms of E(W), a value which can be computed using the G/G/1 formula.

Rewriting this formula in terms of the flow line terminology results in the following estimator of the output variability for a resource:

$$Var(R_i) = \sigma_{a_i}^2 + 2\sigma_{m_i}^2 - \frac{2}{\lambda}(1-\rho_i)E_{G/G/1}[T_i] \qquad (4.2)$$

where: $Var(R_i)$ Variability of the output process of resource R_i (i = 1,...,N)

$\sigma_{a_i}^2$ Variability of the arrival process to resource R_i (i = 1,...,N)

$\sigma_{m_i}^2$ Variability of resource R_i's service time (i = 1,...,N)

ρ_i Traffic intensity at resource R_i: $\rho_i = \frac{\lambda}{\mu_i}$ (i = 1,...,N)

λ Arrival rate to the flow line

μ_i Average service rate at resource R_i (i = 1,...,N)

$E_{G/G/1}[T_i]$ Expected waiting time for resource R_i (i = 1,...,N) using the G/G/1 formula. The cycle time for Resource 1 (*i.e.*, R_1) is computed using the M/G/1 formula.

Therefore, for a single server flow line, the cycle time of R_1 is computed with the M/G/1 queuing formula while subsequent queues use the G/G/1 formula. The mean time between arrivals is the same as the arrival mean to R_1, but the variability must be estimated with equation (4.2) and must be computed sequentially through the flow line.

To illustrate this extended procedure for computing the cycle time of a resource consider the following example (Figure 4.1) of a three resource flow line, which consists of only single capacity servers.

The data describing this example is summarized in Table 4.1. Note that values for the variability of the arrival to the queue of each resource (v_{i-1}) have been computed. For instance, with the mean time between part arrivals being 100, v_o, the variability of the arrival process to Q_o of R_1 is 100^2, since arrivals to the first resource are assumed to be exponential.

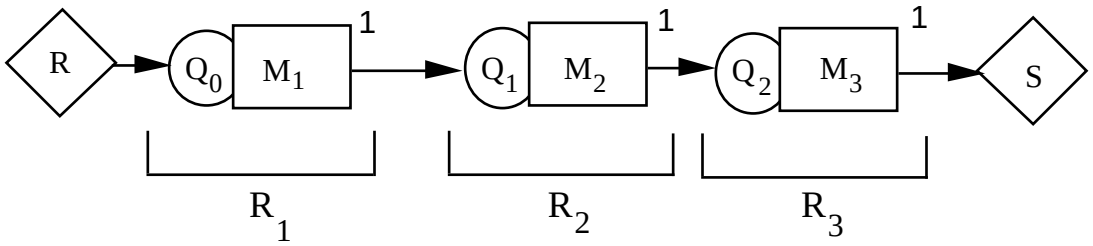


Figure 4.1. Example of a three resource, single server capacity flow line manufacturing system.

Table 4.1. Definition of the single server flow line.

$$FL = \langle R, R_1, R_2, R_3, S \rangle$$
$$R = \langle 100, \infty \rangle$$
$$S = \langle \infty \rangle$$

$R_1 = \langle Q_O, M_1 \rangle$	$R_2 = \langle Q_1, M_2 \rangle$	$R_3 = \langle Q_2, M_3 \rangle$
$Q_O = \langle 100^2, \infty \rangle$	$Q_1 = \langle 3608.33, \infty \rangle$	$Q_2 = \langle 3358.62, \infty \rangle$
$M_1 = \langle Uniform(75,85), 80, 1 \rangle$	$M_2 = \langle Triangular(32,43,60), 45, 1 \rangle$	$M_3 = \langle Uniform(64,80), 72, 1 \rangle$

With all the necessary data for R_1, its cycle time (T_1) can be computed, so that the variance of the output process can be estimated:

$$T_1 = \frac{\lambda\left(m_1^2 + \sigma_{m_1}^2\right)}{2\left(1-\rho_1\right)} + m_1 = \frac{1/100\left(80^2 + 8.3333\right)}{2(1-.8)} + 80 = 240.2083$$

The variability of the arrival process to R_2, v_1, can be estimated by equation (4.2), which computes the variability of resource R_1's output process:

$$\begin{aligned} v_1 = Var(R_1) &= \sigma_{a_1}^2 + 2\sigma_{m_1}^2 - \frac{2}{\lambda}(1-\rho_1)E\left[T_1\right] \\ &= 100^2 + 2(8.3333) - \frac{2}{.01}(1-.8)(240.2083 - 80) \\ &= 3608.3333 \end{aligned}$$

Thus, the arrival process to R_2 has a mean of 100 (since the mean time between arrival remains constant throughout the flow line) and a variance of 3608.33333. Therefore, the squared coefficient of variation of the arrival variation is $3608.33333/100^2 = .360833$. Using the G/G/1 formula, the expected cycle time of R_2 can be estimated:

$$\begin{aligned} T_2 &= \left\lfloor \frac{cv_{a_2}^2 + cv_{m_2}^2}{2} g\left\{\frac{\rho_2}{1/m_2 - \lambda}\right\}\right\rceil + m_2 \\ &= \left\lfloor \frac{.360833+.0163786}{2}(.413758)\left\{\frac{.45}{.02222-.01}\right\}\right\rceil + 45 \\ &= 47.8732 \end{aligned}$$

With an estimate of the cycle time for R_2, the arrival variability to R_3 (output variability of R_2) can be computed:

$$\begin{aligned} v_2 = Var(R_2) &= \sigma_{a_2}^2 + 2\sigma_{m_2}^2 - \frac{2}{\lambda}(1-\rho_2)E\left[T_2\right] \\ &= 3608.3333 + 2(33.1667) - \frac{2}{.01}(1-.45)(47.8732 - 45) \\ &= 3358.62 \end{aligned}$$

Thus, the arrival process to R_3 has a mean of 100 and a variance of 3358.62. The squared coefficient of variation of the arrival variation is $3358.62/100^2 = .335862$. Using the G/G/1 formula, the expected cycle time of R_3 can be estimated:

$$\begin{aligned} T_3 &= \left\lfloor \frac{cv_{a_3}^2 + cv_{m_3}^2}{2} g\left\{\frac{\rho_2}{1/m_3 - \lambda}\right\}\right\rceil + m_3 \\ &= \left\lfloor \frac{.335862+.00411523}{2}(.714367)\left\{\frac{.72}{1.3889-.01}\right\}\right\rceil + 72 \\ &= 94.4827 \end{aligned}$$

With all cycle times computed, the average cycle time of the aggregation resource, AR_1, can be determined:

$$\overline{T_1^*} = \frac{T_1 + T_2 + T_3}{3}$$

$$\overline{T_1^*} = \frac{240.2083 + 47.8732 + 94.4827}{3} = \frac{382.564}{3} = 127.521$$

The results of determining the average aggregate cycle time are summarized in Table 4.2. The remainder of the aggregation methodology works exactly as before in that the next step is to determine the aggregation resource service mean and then develop the distribution weights. It is hypothesizes that this extension to the aggregation methodology results in an average aggregation resource cycle time of AR_1 which is a better estimate that the case when independence is assumed. Since running the aggregate simulation model attempts to replicate this estimate, the resulting estimate of cycle time should be a good approximation.

Table 4.2. Summary of results for single server example.

	Resource #1	**Resource #2**	**Resource #3**
Arrival Mean	100	100	100
Arrival Variability	10,000	3608.33	3358.62
Est. Cycle Time	240.208	47.8732	94.4827

4.1.3 Upper Bound Estimate of the Expected Cycle Time

In the case of a flow line with resources having general service time distributions (*i.e.*, not all exponential) and service capacity not all one, then applying the aggregation methodology provides the specifications for creating an aggregate simulation model which is an upper bound estimate of the expected cycle time of a part.

Unlike the single server case presented in Section 4.1.2, there are no easy techniques for estimating the variability of the queue output for more than one server. Whitt (1983) does develop a method for estimating a point process (departure) by a renewal process. Unfortunately, this technique is not at a point in where it is available for a general class of queueing systems.

There is a theorem, *Khintchine's Theorem*, which states that the superposition of *n* renewal processes becomes Poisson, when properly rescaled, as *n* goes to infinity. Wolff (1989) remarks that in general it is not useful for obtaining approximations. The reason is that the quality of the approximation from the theorem depend not only on *n* and the interarrival distribution, but also on the application. For instance, if the composition process is the input into a queue, is traffic heavy or light? In light traffic, the approximation only needs to be good for the next few arrivals. For heavy traffic, the variability of the arrival process over many arrivals is important. Based on this work, Whitt (1984) shows that the output process in a large class of G/G/S systems is approximately Poisson when there are many busy slow servers.

Rather than estimate the output process, this research assumes resource are independent of one another and that order has no impact on estimating the cycle time. The cycle time of all resources is computed with the M/G/S queueing formula, since each could be the first resource in the flow line (which by definition has a Poisson arrival pattern). Thus, the problem of estimating the shift or change in the variability has been removed.

Running the aggregate simulation model provides an upper bound estimate of the expected cycle time. To understand, realize that the arrival process to a resource is probably not Poisson. Thus, the cycle time of a resource should be computed with a G/G/S formula (presented in Section 2.3.3). But this formula requires an estimate of the arrival variability, which is unknown and cannot be easily estimated. It is assumed to be

Poisson when in fact it probably is not. Therefore, by using a M/G/S estimates of a resource's cycle time (Section 3.2) in place of a G/G/S estimate, the aggregation resource cycle time is being overestimated.

To show this, first it is necessary to show that the coefficient of variation of the output process of an M/G/S queue is less than or equal to one given the restriction that the service time coefficient of variation is less than or equal to one. This is stated in Assertion 4.1. The assertion shows this true for the single server case and it will be assumed that it holds valid for any capacity resource. The assertion is written in the perspective of the output process, cv_{τ_n}. This quantity is equivalent to cv_a, the coefficient of variation of the arrival process. The only difference between these two is their perspective to the queue, in that one addresses the arrival coefficient of variation and the other address the output.

Assertion 4.1 If the service time coefficient of variation of a resource is less than one (as defined in Chapter 3), then the coefficient of variation of the resource (queue) output must also be less than or equal to one. That is, if $cv_{\bar{t}} \leq 1$, then $cv_{\tau_n} \leq 1$. Correspondingly, this can also be written as if $cv_{\bar{t}} \leq 1$, then $cv_a \leq 1$.

Discussion: By assuming that $cv_{\tau_n} > 1$, a contradiction will be reached, thus showing $cv_{\tau_n} \leq 1$. Section 4.1.2 showed that the variance of τ_n, the interdeparture interval between the nth and (n+1) departures variance of the output process is:

$$\sigma_{\tau_n}^2 = \sigma_{\bar{t}}^2 + \frac{1-\rho^2}{\lambda^2}$$

Multiplying λ^2 to both sides results in:

$$\lambda^2\sigma_{\tau_n}^2 = \lambda^2\sigma_{\bar{t}}^2 + 1 - \rho^2$$

But, $cv_{\tau_n} = \lambda\sigma_{\tau_n}$ and $cv_{\bar{t}} = \lambda\sigma_{\bar{t}}$. Thus, $cv_{\tau_n}^2 = \lambda^2\sigma_{\tau_n}^2$ and $cv_{\bar{t}}^2 = \lambda^2\sigma_{\bar{t}}^2$.

Replacing these yields:

$$cv_{\tau_n}^2 = \lambda^2 \sigma_{\bar{t}}^2 + 1 - \rho^2$$

Assuming that $cv_{\tau_n} > 1$:

$$cv_{\tau_n}^2 = \lambda^2 \sigma_{\bar{t}}^2 + 1 - \rho^2 > 1$$

Thus:

$$\lambda^2 \sigma_{\bar{t}}^2 - \rho^2 > 0$$

$$\lambda^2 \sigma_{\bar{t}}^2 > \rho^2$$

But $\rho = \lambda \bar{t},$ thus $\rho^2 = \lambda^2 \bar{t}^2$, where $\bar{t}$ is the service mean. Replacing yields:

$$\lambda^2 \sigma_{\bar{t}}^2 > \lambda^2 \bar{t}^2$$

Since by definition $0 \le \lambda \le 1,$ thus $0 \le \lambda^2 \le 1$. Multiplying both sides by $1/\lambda^2$ yields:

$$\sigma_{\bar{t}}^2 > \bar{t}^2$$

Since $\bar{t}^2$ is non-negative (there cannot be non-negative service time):

$$\frac{\sigma_{\bar{t}}^2}{\bar{t}^2} > 1$$

But, $cv_{\bar{t}}^2 = \frac{\sigma_{\bar{t}}^2}{\bar{t}^2}$, thus:

$$cv_{\bar{t}}^2 > 1 \text{ and } cv_{\bar{t}} > 1$$

Hence, by assuming that $cv_{\tau_n} > 1$ it has been shown that $cv_{\bar{t}} > 1$. But, by definition $cv_{\bar{t}} \le 1$. Thus, a contradiction has occurred. Therefore, $cv_{\tau_n} \le 1$.

Assertion 4.1 demonstrates that if the coefficient of variation of the service time is restricted to be less than one (which it is defined to be), then the coefficient of variation of the output (cv_{τ_n}) from a queue in the flow line must also be less than or equal to one. This assertion shows this for the single server case. The variance of the multiple server case is unknown, but it will be hypothesized that it is also less than one.

This fact can be used in combination with the upper bound estimate of the expected waiting time (cycle time) in a G/G/S queue (Section 3.3.3) to show that the a M/G/S queue overestimates a G/G/S queue:

λ	Mean arrival rate
σ_a^2	Variance of the interarrival time
μ	Mean service rate
$\bar{t}$	Mean service time
$\overline{\bar{t}^2}$	Second moment of the service time
$\sigma_{\bar{t}}^2$	Variance of the service time
S	Number of parallel, identical servers
ρ	Traffic intensity: $\rho = \frac{\lambda}{S\mu}$
$E_{G/G/S}[W]$	Expected waiting time for G/G/S queue

$$E_{G/G/S}[W] \le \frac{\sigma_a^2 + (1/S)\sigma_{\bar{t}}^2 + \left[\frac{(S-1)}{S}\right]\bar{t}^2}{2\bar{t}(1-\rho)} + \bar{t}$$

If the arrival process is Poisson (the variability of the interarrival time is equal to $1/\lambda^2$) the following upper bound estimate for the expected waiting time of a M/G/S queue is:

$$E_{M/G/S}[W] \le \frac{\left(\frac{1}{\lambda^2}\right) + (1/S)\sigma_{\bar{t}}^2 + \left[\frac{(S-1)}{S}\right]\bar{t}^2}{2\bar{t}(1-\rho)} + \bar{t}$$

$E_{M/G/S}[W]$ will always be greater than $E_{G/G/S}[W]$ when $\frac{1}{\lambda^2} \ge \sigma_a^2$. This simplifies to: $\lambda^2\sigma_a^2 \le 1$. But, $\lambda^2\sigma_a^2 = cv_a^2$, where cv_a^2 equals the squared coefficient of variation of the arrival time. Thus, $E_{M/G/S}[W]$ is an upper bound to $E_{G/G/S}[W]$ when $cv_a^2 \le 1$ or correspondingly, when $cv_a \le 1$. But, $cv_{\bar{t}} \le 1$ by definition (Assertion 4.1 showed this fact). Therefore, $E_{M/G/S}[W]$ is an upper bound estimate of $E_{G/G/S}[W]$.

To summarize, by restricting the service time coefficient of variation to be less than or equal to one, Assertion 4.1 showed that the coefficient of variation of the output process (arrival process) must also be less than or equal to one. Hence, the exponential distribution which has a coefficient of variation of one is the largest variation that satisfies this requirement. Intuitively this makes since considering the statement: *the more varied the arrival pattern, the longer the waiting time* (Hendricks, 1992). Thus, by modeling the G/G/S resource with an M/G/S queueing estimate provides an upper bound estimate of the expected queue waiting time.

4.2 Computer Implementation of Aggregation Methodology

The aggregation methodology as presented in Chapter 3 and the single server extension presented in Section 4.1.2 have been implemented into a computer program using the Mathematica programming language. Figure 4.2 describes the relationship between the different program modules. Table 4.3 summarizes each of the program modules and describes its function. A listing of the complete program is presented in Appendix B.

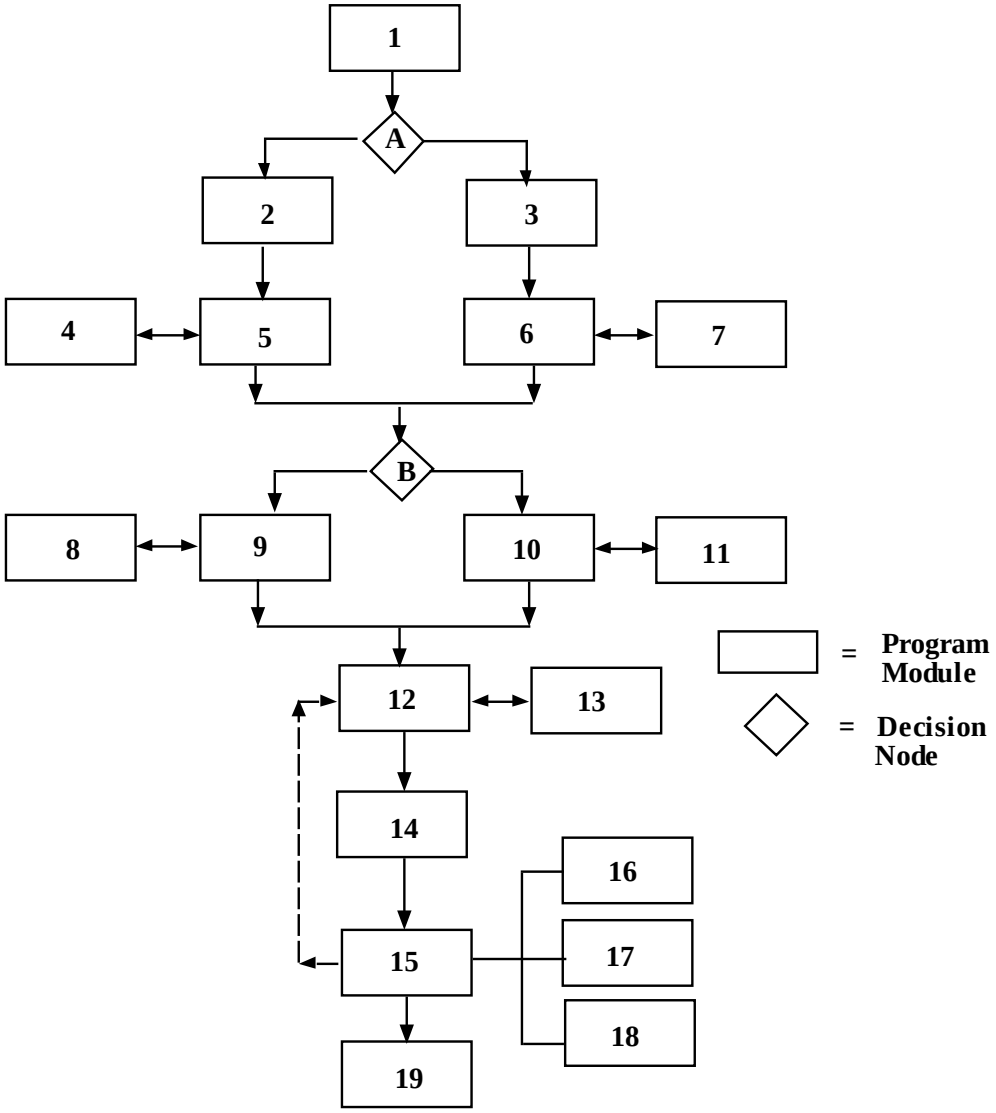


Figure 4.2. Relationship between aggregation program modules.

Table 4.3. Definition of the modules in the computer implementation of the aggregation methodology.

Module	Purpose
1	Aggregation - main program module, controls overall execution.
2	GenerateMain - main module for controlling the generation of a flow line.
3	UserMain - main module for controlling a user inputted flow line.
4	Step1Generate - generates the flow line description.
5	enterGenServer - generates the number of servers for a resource.
6	Step1 - inputs the flow line description.
7	enterserver - inputs the number of servers for a resource.
8	GG1CycleTime - computes the cycle time of a resource using G/G/1 formula.
9	Step2 - control the computation of a resources cycle time.
10	Step2Single - controls the computation of a resources cycle time for a single server flow line.
11	MGSCycleTime - computes the cycle time of a resource using M/G/S formula.
12	Step3 - controls the computation of the service mean for an aggregation resource.
13	MGSServiceMean - computes the service mean for an aggregation resource.
14	FPrint - prints a description of each resource in the flow line.
15	Step4 - controls the computation of the distribution weights
16	Swap - stores and orders values before solving for distribution weights.
17	DistWeight1 - recursive algorithm for determining the distribution weights composing an aggregation resource service mean using a pairwise aggregation.
18	DistWeight2 - recursive algorithm for determining the distribution weights composing an aggregation resource service mean using a consecutive aggregation.
19	APrint - prints the characteristics of the aggregation resources, including the distribution weights.

A	Decision Node - enter flow line description or have the computer randomly generate a test case
B	Decision Node - only single servers or multiple servers

Once started, the program initially ask the user whether a flow line is to be manually entered or whether one is to be randomly generated. By selecting the manual input option, the user specifies whether the flow line has only single capacity servers or not. Next, the user specifies the time between the arrival of parts to the flow line. The last task for the user is to sequentially enters each resource of the flow line by selecting its service distribution from a menu and specifying the its distributions parameters and the number of parallel servers. Available service distributions include:

- Exponential
- Lognormal
- Normal
- Triangular
- Uniform

Once this information is entered, the aggregation program computes the cycle time for each resource. The procedure for accomplishing this depends upon whether or not the flow line has only single capacity resources. If it does, the extension of Section 4.1.2 for estimating the arrival variability is used, otherwise the procedure of Section 3.2 is applied. Next, the average aggregate resource cycle time is computed. Each of the aggregate resource cycle times are used to compute the aggregate resource service mean necessary for creating a resource with the given cycle time and arrival mean.

Once all aggregate service means have been computed, the program prints out a description of each resource in the flow line. This description includes such information as each resource's arrival mean, arrival rate, arrival variability, arrival coefficient of variation, and arrival squared coefficient of variation. It also specifies the number of parallel resources for the resource, the service distribution and its parameters, the service mean, the service variability, the service time standard deviation, the service time coefficient of variation, the service time squared coefficient of variation, and the resource

utilization. In addition, the estimated queue waiting time for the resource is included along with the resource's estimated cycle time.

The next task of the aggregation program is to compute the resource distribution weights for each of the aggregation resources. This is done using the recursive algorithm presented in Section 3.4. Once all the resource distribution weights have been computed, the program prints out a description of each aggregation resource specifying its arrival mean, arrival rate, total aggregate cycle time, average aggregate cycle time, estimated service time squared coefficient of variation (determined with the weighting procedure of Chapter 3), and the estimated aggregation resource service mean and rate. Also, the distribution weights for each of the resources of the aggregation resource are specified.

Appendix C provides the program generated description from entering the MPD manufacturing flow line (described throughout Chapter 3). This flow line is one with multiple capacity servers. Hence, the arrival to each resource is assumed to be independent, with a mean rate of 100. Note that the listing (of Appendix C) indicates that the arrival variability to each of the six resources of the flow line is 10,000 (100^2). In comparison, Appendix D presents the program generated description for the single server flow line described in Section 4.1.2. In this system, the arrival variability is computed for resources two and three of the flow line.

If the user initially selects the computer generated flow line option, the program prompts the user to specify whether it should generate an exponential test case (all exponential servers), a single server test case (all single capacity servers), or a multiple server case (multiple servers). Once a choice is made, the program randomly generates the number of resources in the flow line. For an exponential or multiple server test case, the number of resources is an integer between 5 and 20. Thus, a test case consists of between 5 and 20 resources.

It should be noted that a single server test case consists of between 5 and 10 resources. In this type of system, all resources are aggregated together to form a single aggregation resource, namely AR_1. In (say) a ten resource single server flow line, all ten resource means must be weighted to find their percentage contribution towards the aggregate resource service mean. The procedure to find these weights is to use a recursive algorithm to aggregate within an aggregation resource. Experimentation has shown that with an aggregation resource that represents more than ten resources, applying the recursive algorithm significantly reduces the mean values of the resources. The result of this drastic reduction is that solving the resulting two equations of two unknowns (representing the percentage weight of the two resources represented by the aggregation resource) yields a negative value (since both means are less that the mean they are trying to estimate). For example, $2x + 3y = 5$ and $x + y = 1$ will yield a negative value for one of the factors.

Intuitively this makes since in that there should be a limit to the number of times something can be aggregated before it becomes insignificant. This limit appears to be ten resources. To deal with this limitations, the aggregation program uses twelve solve techniques for finding the distribution weights (using modules DistWeight1 and DistWeight2 in combination with the Swap module). The program initially attempts to find the distribution weights using module DistWeight1 (which aggregates successive resources together). For instance, it aggregates R_1 and R_2 into $R_{1|2}$, and aggregates R_3 and R_4 into $R_{3|4}$. If any of the resulting distribution weights are negative, the program calls the Swap Module, which sorts the resources using one of five different techniques (high to low, low to high, alternating high-low, etc.) and attempts to solve for the weight using DistWeight1. If all six attempts fail, then the aggregation program follows a similar procedure but uses the recursive algorithm of DistWeight2 (which successively aggregates resources - as demonstrated in Section 3.4). For example, it aggregates

resource R_1 and R_2, then resource $R_{1|2}$ with R_3, etc. Experimentation has shown that for the majority of systems with ten or fewer servers, one of the twelve solution techniques finds a solution.

For each resource of any type of flow line system, an estimate of the resource's utilization is generated as a values between ten percent and ninety percent. For the exponential and multiple server test cases, the number of resource servers (from one to eight) is also generated. For the single server case, this value is obviously one. The next component to be generated is the service distribution for the resource. For an exponential test case, the exponential distribution is always selected.

To determine the parameters for each of the service distribution requires differing techniques. For the *exponential* service distribution, the only parameter which needs to be estimated is its mean. Using the relationship that a resource's utilization (*i.e.*, ρ) is equal to its arrival rate multiplied by the service mean divided by the number of servers, the service mean can be determined since the time between arrival (defined to be 100) and the utilization and number of servers are all known.

For the *lognormal* and *normal* service distributions, the mean value is generated using the service time utilization relationship. For simplicity, the standard deviation is specified as a number between one and ten percent of the generated mean value.

To generate a value from the *triangular* distribution requires estimating three parameters: a maximum, a mode, and a minimum. To estimate these, a maximum mean value is computed using the utilization relationship. The program then randomly generates a maximum value, a mode (a value between zero and the maximum), and a minimum (a value between zero and the mode). The program computes the mean for these three parameter estimates. If this mean is less than the originally generated maximum mean, then these are the parameters of the triangular distribution. Otherwise, the program generates three more estimates of the maximum, mode, and minimum. This

procedure continues until the mean of the set of parameters is less than the maximum mean.

For the *uniform* distribution, a mean and standard deviation are computed following the procedure of how the lognormal and normal distributions parameter values were generated. To generate the parameter values (a minimum and a maximum) for the uniform distribution, the following relationship is used (Pritsker, 1986):

$$\text{minimum} = \text{mean} - \sqrt{3} \times (\text{standard deviation})$$

$$\text{maximum} = \text{mean} + \sqrt{3} \times (\text{standard deviation})$$

to allow both of the parameter values to be computed.

The randomly generated components necessary for computing a test case are summarized in Table 4.4. It was decided that these parameter ranges create a set of generated test cases that represent a diverse set of flow line systems. Once all the data for the flow line has been generated, the remaining parts of the aggregation program (computing cycle times, finding distribution weights, etc.) operate exactly the same as before.

Table 4.4. Range of generated parameters for each test case type.

	Exponential	**Single**	**Multiple**
Number of Resources	5 - 20	5 - 10	5 - 20
Resource Utilization	10% - 90%	10% - 90%	10% - 90%
Number of Servers	≤ 8	1	≤ 8
Service Distributions	Only expon.	Any	Any

4.3 Testing the Aggregation Methodology

The aggregation methodology was tested for three types of flow line systems: a flow line with all exponential service time, a flow line of only single server capacity, and a flow line of multiple servers of any service capacity. For each of the three types of systems, ten test cases were generated using the aggregation program presented in Section 4.2. Testing involved developing and running a total of fifty simulation models on a computer.

The simulation models were written in the SLAM simulation language and run on a SUN SparcCenter 2000 in the College of Engineering and Applied Science at Arizona State University. The goal was to compare the estimated cycle time achieved by the aggregate simulation model to that of the full flow line model (or to the analytical steady state results which can be computed for the exponential system). To ensure accurate results, each of the fifty models was run thirty times with the terminating condition that 250,000 parts pass through the flow line. Given a default arrival rate of 100 (which is assumed by the program generator), this equates to each run being approximately 25,000,000 (simulation) minutes in length.

The total run time (time required of the computer central processing unit) for all fifty models was in excess of 180 hours, or equivalently, $7\frac{1}{2}$ days. Simplistic estimates indicate the SUN SparcCenter is eighteen times faster that a 386 (16 mHrtz) personal computer belonging to the System Simulation Laboratory. To obtain the same simulation data with the personal computer would of required in excess of $7\frac{1}{2}$ x 18 = 135 days.

During each model's execution, the statistical arrays are not cleared. Experimentation showed that by requiring 250,000 parts to pass through the flow line system reduced the initial simulation transient and completely negates its impact on the simulation results.

Section 4.3.1 will discuss the results of applying the aggregation methodology to the exponential test cases. Section 4.3.2 reviews the single server results. Section 4.3.3 discusses the multiple server results.

4.3.1 Results for the Exponential Test Cases

To test the effectiveness of applying the aggregation methodology to a flow line with all exponential service times, ten test cases were generated by the aggregation program. Their description is presented in Appendix E. Included is all information necessary for creating the full and aggregate simulation models.

To judge how well the aggregation methodology works for this type of flow line, the aggregate simulation models results are compared to the analytical steady state results (which can be computed for an exponential system). The results are compared in Table 4.5. The results indicate that the average relative error,

$$RE = 100\% \times \left\lceil \frac{\left| \text{Average aggregate cycle time} \;-\; \text{steady state estimate} \right|}{\text{steady state estimate}} \right\rceil$$

associated with the aggregate estimate of the cycle time is only 1.1390%. A 95% confidence interval computed on the average relative error or the cycle time resulting from applying the aggregation methodology to ten exponential flow line systems is: (.5955%, 1.6825%).

Table 4.5. Comparison of average cycle time of the aggregate results to the steady state results for the exponential test cases.

Test Case	Average Cycle Time (Aggregate Model)	Steady State Estimate of Cycle Time	Relative Error
1	3998.93	3964.1910	0.8763%
2	3631.20	3558.4360	2.0448%
3	7501.87	7402.9133	1.3367%
4	1433.77	1433.6360	0.0093%
5	6312.80	6255.3200	0.9189%
6	2946.87	2926.5100	0.6959%
7	7468.73	7349.5300	1.6219%
8	7501.87	7402.9133	1.3367%
9	2202.83	2182.5710	0.9282%
10	7468.73	7349.5300	1.6219%
		Average =	1.1390%
		SD =	.5773%

In an attempt to explore the distribution of simulation output, Table 4.6 summarizes the difference between the *coefficient of variation* (the ratio of the standard deviation compared to the average cycle time) for the aggregate and full simulation models.

Table 4.6. Difference in the coefficient of variation for each of the exponential test cases.

	Full Simulation Model			**Aggregate Simulation Model**			
Test Case	Ave. Cycle Time	SD	COV	Ave. Cycle Time	SD	COV	Diff.
1	4000	28.7	.00717500	3940	38.4	.00974761	-.00257119
2	3630	33.8	.00931129	3550	59	.01661972	-.00719202
3	7500	59.2	.00789333	7420	84	.01132075	-.00337208
4	1430	2.03	.00141958	1430	1.55	.00108392	.00032867
5	6310	26.44	.00419017	6250	42.2	.00675200	-.00251956

6	2950	8.87	.00300678	2920	11.1	.00380137	-.00077694
7	7470	48.2	.00645248	7360	52.9	.00718750	-.00071984
8	6590	33.3	.00505311	6500	49.3	.00758462	-.00249947
9	2200	5.79	.00263182	2180	5.48	.00251376	.00011847
10	8300	93.2	.01122892	8180	166	.0202934	-.00887846

Consider the results for test case 1. The coefficient of variation of the cycle time from running the full simulation model (rather than the steady state estimate) is .007175. That is, the standard deviation .7175% of the mean cycle time. For the corresponding aggregate simulation model, the coefficient of variation is .00974619, or .9746% of the mean cycle time. The difference is .00257119, or .2%. An average difference for all test cases is computed to be .00285407, or .2854%. That is, the standard deviation for the full model and aggregate model estimate of the cycle time differ by and average of .2854%. Thus, it appears that the variability of the output distribution generated by the aggregate and full model are similar.

4.3.2 Results for the Single Server Test Cases

To test the single server extension presented in Section 4.1.2, ten test cases were generated and modeled. This involved running twenty simulation models (a full model and it corresponding aggregate models for each test case description). A description of these test cases is included in Appendix F. The results are summarized in Table 4.7.

The results indicate that the average relative error,

$$RE = 100\% \times \left\lceil \frac{\left| \text{Average aggregate cycle time} - \text{Average Full Model Cycle Time} \right|}{\text{Average Full Model Cycle Time}} \right\rceil$$

of the aggregate's estimate of the cycle time is 4.8735%. A 95% confidence interval computed on the average relative error or the cycle time resulting from applying the aggregation methodology to ten single server flow line systems is: (4.3333%, 5.4137%).

Table 4.7. Comparison of average cycle time of the aggregate results to the full model simulation results for the single server test cases.

Test Case	Average Cycle Time (Aggregate Model)	Average Cycle Time (Full Model)	Relative Error
1	284.340	266.060	6.8706%
2	341.947	313.707	9.0020%
3	957.280	938.357	2.0166%
4	684.470	654.050	4.6510%
5	612.100	590.790	3.6070%
6	677.427	676.797	0.0931%
7	691.490	643.930	7.3859%
8	214.980	203.470	5.6569%
9	258.057	23.940	7.5506%
10	1012.400	993.5100	1.9013%
		Average =	4.8735%
		SD =	2.9179%

As was done with the exponential test cases, an analysis of the variability of the output distribution is presented in Table 4.8.

Table 4.8. Difference in the coefficient of variation for each of the single server test cases.

	Full Simulation Model			**Aggregate Simulation Model**			
Test Case	Ave. Cycle Time	SD	COV	Ave. Cycle Time	SD	COV	Diff.
1	284	.779	.00274296	266	.436	.00163910	.00110386
2	342	.630	.00184211	314	.525	.00167197	.00017013
3	957	8.80	.00919540	938	6.84	.00729211	.00190329
4	684	4.32	.00631579	654	3.20	.00489297	.00142282
5	612	2.8	.00457516	591	1.81	.00306261	.00151256
6	677	1.48	.00218612	577	1.49	.00220089	-.00000147
7	691	3.32	.00480463	644	3.14	.00487578	-.00000711
8	215	.512	.00238140	203	.234	.00115271	.00122869
9	258	.529	.00205039	240	.390	.00162500	.00042539

10	1010	6.04	.0059802	994	7.4	.00744467	-.00146447

For the single server flow line system, the average difference in the variation for all test cases is computed to .00062164, or .0621%. That is, the standard deviation for the full model and aggregate model estimate of the cycle time differs by an average of .06%. Thus, it appears that the variability of the output distribution generated by the aggregate and full model is similar for the single server system.

4.3.3 Results for the Multiple Server Test Cases

For the multiple server system, ten test cases (flow lines with multiple servers of any general distribution) were generated and tested (see Appendix G). For each test case, the full and corresponding aggregate simulation model were run. As outlined in Section 4.1.3, it is hypothesized that for this type of system, due to the independence assumption, the aggregate results should provide an upper bound to the average cycle time. The results are summarized in Table 4.9.

Table 4.9. Comparison of average cycle time of the aggregate results to the full model simulation results for the multiple server test cases

Test Case	Average Cycle Time (Aggregate Model)	Average Cycle Time (Full Model)	Relative Error
1	3591.57	3339.00	7.5642%
2	3735.00	3492.00	6.9382%
3	2496.17	2438.67	2.3578%
4	1078.50	1042.33	3.4701%
5	2707.87	2701.70	0.2284%
6	2519.67	2315.73	8.0670%
7	1320.97	1279.70	3.2250%
8	2287.40	2278.80	0.3774%
9	1468.13	1458.60	0.6534%
10	2436.37	2367.47	2.9103%
		Average =	3.5792%
		SD =	2.9706%

In terms of the relative error,

$$RE = 100\% \times \left\lceil \frac{\left| \text{Average aggregate cycle time} - \text{Average Full Model Cycle Time} \right|}{\text{Average Full Model Cycle Time}} \right\rceil$$

the aggregate cycle time overestimated by an average of a little more than $3\frac{1}{2}$ percent. As expected, the average aggregate cycle time is always larger than the full model estimate.

The results are quite interesting considering in that they are very close even though the multiple server case assume independence among resources and models each with a Poisson arrival process. A possible explanation follows the discussion of Section 4.2.3 in which Whitt remarks that in a G/G/S queueing system with many busy servers, the output process of a resource tends to be exponential. These results indicate that this is in fact true.

As was done with the previous two types of test cases, an analysis of the variability of the output distribution is presented in Table 4.10.

Table 4.10. Difference in the coefficient of variation for each of the multiple server test cases.

	Full Simulation Model			**Aggregate Simulation Model**			
Test Case	Ave. Cycle Time	SD	COV	Ave. Cycle Time	SD	COV	Diff.
1	3590	3.8	.00105850	3340	2.92	.00087425	.00018424
2	3740	5.29	.00141444	3490	4	.00114613	.00026831
3	2500	4.05	.00162000	2440	3.1	.00127049	.00034951
4	1080	2.21	.00204630	1040	1.9	.00182692	.00021937
5	2710	2.33	.00085978	2700	2.02	.00074815	.00011163
6	2520	3.26	.00129365	2320	2.13	.00091810	.00037555
7	1320	1.4	.00106061	1280	.596	.00046563	.00059498
8	2290	2.01	.00087773	2280	1.56	.00068421	.00019352
9	1470	.776	.00052789	1460	.724	.00049589	.00000320
10	2440	2.24	.00091803	2390	1.87	.00078243	.00013561

For the multiple server flow line system, the average difference in the variation for all test cases is computed to .00024647, or .02467%. That is, the standard deviation for the full model and aggregate model estimate of the cycle time differ by an average of .02467. Thus, it appears that the variability of the output distribution generated by the aggregate and full model is similar for the multiple server system.

4.4 Chapter Summary

This chapter illustrates the application of the aggregation methodology. It achieves this by:

(1) Dividing the application of the aggregation methodology into three types of systems. The first is a pure exponential system in which all service distributions are exponentially distributed with any number of parallel, identical servers allowed. It is hypothesized that the aggregation methodology will work well for this system since the exact steady state results are used by the aggregation methodology for creating the aggregate simulation model. The second type of system is a single server capacity flow line, in which each resource of the flow line has one server performing the resource's service task. There is no restriction of the type of service distribution other than the coefficient of variation of the service time must be less than or equal to one. To model this system requires the aggregation methodology to be extended such that the arrival variability to a resource is estimated. The final type of system is one with multiple servers of any service distribution (also with the restriction that the service time coefficient of variation is less than or equal to one). It is hypothesized that in this system, the average cycle time estimate produced by running the aggregate simulation model will be an upper bound estimate of the average cycle time for the full (or true) system.

(2) Explains the development of a computer program which implements the aggregation methodology. The program either computes the distributions weights from a user inputted flow line description or randomly generates a test case of one the three types (exponential, single, or multiple).

(3) Compares the effectiveness of each of the three types of systems in estimating the average part cycle time. All three methods produce good estimates in terms of the

relative error of the mean cycle time In comparing the variability of the output distribution for each of the three types of test cases, the resulting error or difference in the variability of the cycle time estimated is minor. This indicates that the output distributions for the full and aggregate simulation model are similar. As hypothesized, for the multiple server system, all test cases show that the aggregate results is an upper bound estimate of the full models estimate of the cycle time.

CHAPTER 5

SUMMARY, LIMITATIONS, CONCLUSIONS, AND RECOMMENDATIONS

The many uses of simulation range from comparing alternative systems to answering capacity and feasibility questions. Unfortunately, the potential benefits that discrete event simulation offers are impeded by the high level of expertise necessary to successfully conduct a sound simulation study. As a solution, this research proposes aggregation techniques for aiding in the development of manufacturing flow line simulation models. Section 5.1 summarizes the general approach of applying the aggregation methodology. Section 5.2 discusses the limitations associated with the aggregation methodology. Section 5.3 reviews the specific achievements of this research and summarizes the effectiveness of applying the aggregation techniques to three types of flow line systems. Section 5.4 concludes the discussion by discussing future research work on this topic.

5.1 Summary of Research Results

The objective of this research was to develop a formal methodology for creating an aggregate simulation model that can be used to estimate part cycle time for a flow line manufacturing system. The resulting aggregation methodology integrates aspects of queueing theory, a recursive algorithm, and simulation to develop the specifications necessary for combining the resources of a flow line so that they can represented by aggregation resources.

The first step of the methodology (Section 3.1) defines the parameters of a production flow line manufacturing system. To collect this information, a mathematical formalism (Table 3.3) describing a flow line system is developed. It provides a foundation for identifying and collecting information about the manufacturing system. From this description, an aggregate representation is developed. This research proposes

that in an aggregate representation of a system, all resources with a given service capacity are aggregated together to form aggregation resources. A flow line system can thus be described by an equivalent aggregate system, in which all same capacity resources are aggregated. Section 3.1.3 develops an aggregate mathematical formalism (Table 3.5) for summarizing all the information describing this equivalent aggregate system.

Once a manufacturing flow line and its aggregate equivalent is described, the average cycle time of an aggregation resource is computed (Section 3.2). Using a M/G/S queueing formula (Section 2.3.2), the cycle time is computed for each resource represented by an aggregation resource. The cycle times of the resources aggregated by an aggregation resource are summed and their average is computed. This value is the average cycle time that the aggregation resource represents. In the special case of a flow line with one single server resources (Section 4.1.2), the cycle time of the resources is computed using a G/G/1 formula (Section 2.3.3) in combination with a procedure for estimating the arrival time variability of resources in the flow line.

With the average aggregate cycle time computed for all aggregation resources, the service mean needed to create an aggregate resource with the given cycle time, arrival rate, and number of servers is computed for each of the aggregate resource (Section 3.3). The procedure for accomplishing this involves applying queueing formulas backwards, in that the mean service time of an aggregation resource is estimated from the average cycle (waiting) time. This differs from the more common approach of specifying the mean service time and computing the cycle (waiting) time.

The next step of the aggregation methodology involves using a recursive algorithm for weighting the resource service time means of the aggregate resource (Section 3.4). These weights represent the percentage contribution of each resource's service mean towards its aggregation resource service mean.

The final step of the aggregation methodology is to specify the aggregate simulation model. This task is complicated by the fact that since general service times are allowed, developing a combined or joint distribution of an aggregation resource service time density function may be neither feasible, efficient, nor possible. The approach of this research is to represent this unspecified service time distribution not as a mathematical function, but rather as a relationship that random numbers can be sampled from (Section 3.5). The resource distribution weights are used in combination with composition sampling (Section 2.1.4) to generate samples from the unspecified aggregate resource service time distribution during the execution of the aggregate simulation model.

5.2 Limitations of the Research

To develop the aggregation methodology required several assumptions to be made. These result in the following limitations in applying the methodology:

(1) Basic flow line system:

The manufacturing flow line assumed for this research is a basic system that requires that all parts pass though the same sequence of resources with no feedback, scrap, or rework. In this system, workers and machines only operate at an assigned resource (station) and do not aid or assist other resources in the system. In addition, it is assumed that resources are one hundred percent reliable and never experience a breakdown.

(2) Single part type:

The aggregation methodology only works for a flow line system producing a single part type. This single part type is assumed to arrive from an infinite supply one at a time according to an exponential distribution. Batch arrivals or batch processing are not considered.

(3) Only part cycle time is estimated:

This research applies the aggregation methodology to a system to estimate a single characteristic of the flow line system, the average cycle time of a part to be processed by all resources in the manufacturing system. While this is an important system characteristic, others such as resource utilization would also be useful. As shown in Section 4.3.1 and Section 4.3.2, for an exponential or single server flow line, an average cycle time error interval must be used to bound the true average cycle time. The disadvantage of this that the range of the error interval is much larger (wider) than a simple confidence interval on the aggregate simulation model mean.

(4) Queues have unlimited storage capacity:

Resources have no set storage capacity. With the assumption that the system is running in steady state, one can be guaranteed that none of the queues grow unbounded. In actuality, queue capacity may be a constraining factor of a system, and thus it may be inappropriate to make this assumption. The ability to account for queue capacity of a resource and the receiving area of the flow line are included in the formalism describing the production flow line (see Table 3.3). At present, this research assumes these values are infinity.

(5) Simplified part selection:

All resources select parts for their associated queue using a first come, first serve (FCFS) selection rule. While this may not necessarily be true in most systems, Section 2.3.4 does address the fact that the impact of the queue selection is minimized in a general queueing system operating under steady state conditions.

(6) An aggregate resource is limited to representing at most ten resources:

As explained in Section 4.2, when more than ten resources are represented by an aggregation resource, the process of recursively determining the distribution weights is quite difficult. In fact, experimentation shows that when more than ten resources are represented by an aggregation resource, the success rate (for determining the distribution weights) is only ten percent or less.

(7) The coefficient of variation of the service time distribution is less than or equal to one:

This requirement is necessary to be assured that the coefficient of variation of the output (departure) process from a queue is also less than or equal to one (shown in Section 4.1.3). While this may be a restrictive assumption in that it excludes certain distributions, it does allows for the modeling of many of the common simulation service time distributions: normal, lognormal, triangular, exponential, uniform.

5.3 Conclusions

This research has made an important first step in applying analytical procedures to the process of developing a simulation model. It demonstrates that analytical techniques such as queueing analysis can be integrated with simulation to reduce the effort necessary to address simulation questions. Specific achievements of this research include:

(1) Development of system formalisms (Table 3.3 and Table 3.5) for describing a production flow manufacturing system and its aggregate equivalent (Section 3.1). From the description of the "as is" system, an aggregate description can be developed by aggregated all same capacity resources into aggregation resources. The two formalisms provide a necessary foundation for identifying and collecting information for flow lines to be defined and compared.

(2) Identification of procedures for computing the average cycle time of an aggregate resources (Section 3.2 and Section 4.1.2). For a flow line system with all exponential servers or one with multiple servers with any service time distributions, this is accomplished by computing the average cycle time of all resources aggregate by an aggregation resource. The individual resource cycle time estimates are computed using a M/G/S queueing formula (Section 2.3.2). For a flow line in which all resources have only single capacity servers, a G/G/1 (Section 2.3.3) queueing formula is used in conjunction with a technique for estimating the arrival (output) variability to a resource in the flow line (Section 4.1.2) for estimating the average aggregate cycle time.

(3) Development of a technique for estimating the mean service time of an aggregate resource (Section 3.3). The procedure involves applying queueing formulas backwards, in that the mean service time of an aggregation resource is estimated from the average cycle (waiting) time.

(4) Creation of a method for describing the mean service time of an aggregation resource in terms of the resource service means that it represents (Section 3.4). Using a recursive algorithm (Table 3.9), the percentage contribution (or distribution weight) of each resource's service mean towards the aggregate resource service mean is computed for each aggregation resource.

(5) Specifications for creating an aggregate simulation model (Section 3.5). The key to this model development involves combining the distribution weights with composition sampling (Section 2.1.3) to sample from the unspecified aggregate resource service time distribution during the execution of the aggregate simulation model. Appendix A presents an example flow line. It contain the SLAM simulation code for the full simulation model and its aggregate equivalent.

(6) Creation of a computer program which implements the aggregation methodology (Section 4.2). The program (Figure 4.2, Table 4.3, Appendix B), written in the Mathematica programming language, computes the resource distribution weights from a user inputted flow line description or for a randomly generated test case (a pure exponential flow line, a single server flow line, or a multiple server flow line). Parameter ranges used by the program for generating a test case are summarized in Table 4.4. Examples of the output generated by the aggregation program are listed in Appendix C and D.

(7) Testing of the aggregation methodology on three types of systems to compare its effectiveness in estimating the average part cycle time (Section 4.3) :

   (a) The first system that was studied is a pure exponential system in which all service distributions are exponentially distributed with any number of parallel, identical servers allowed (Section 4.1.1). It is hypothesized that the aggregation methodology will work well for this system since the exact steady state results are used by the aggregation methodology for creating the

aggregate simulation model. Testing (Section 4.3.1) on ten randomly generated diverse test cases (listed in Appendix E) results in a relative error between the average aggregate estimate of cycle time and the true steady state solution (which can be computed for an exponential system) as being 1.139% (Table 4.5). A 95% confidence interval computed on the average relative error or the cycle time resulting from applying the aggregation methodology to ten exponential flow line systems is: (.5955%, 1.6825%). In comparing the variability of the output distribution, the resulting error or difference in the variability of the cycle time estimated is on average .2854%.

(b) The second type of flow line that the aggregation methodology was applied to is a single server capacity flow line. This type of flow line has one server performing each of the resource's service task with no restriction of the type of service distribution other than the requirement that the coefficient of variation of the service time be less than or equal to one. To model this system requires the aggregation methodology to be extended such that the arrival variability to a resource is estimated (Section 4.1.2). Ten test cases (Appendix F) were generated and the results indicate an average relative error of 4.8735%. A 95% confidence interval computed on the average relative error or the cycle time resulting from applying the aggregation methodology to ten single server flow line systems is: (4.3333%, 5.4137%). In comparing the variability of the output distribution, the resulting error or difference in the variability of the cycle time estimated is on average .06214%.

(c) The final type of system that the aggregation methodology was tested on is one with multiple servers of any service distribution. Assertion 4.2 of Section 4.1.3 showed mathematically that in this type of system the average cycle time estimate produced by running the aggregate simulation model

should be an upper bound estimate of the average cycle time for the full (or true) system. To test this assertion, ten simulation models (Appendix G) were generated by the aggregation program and the corresponding full and aggregate simulation models were modeled. The results (Table 4.9) indicate a relative error between the average aggregate cycle time and the average full model cycle time of only 3.5%. In comparing the variability of the output distribution, the resulting error or difference in the variability of the cycle time estimated is on average a low .0246%. As hypothesized, for the multiple server system, all test cases show that the aggregate results is an upper bound estimate of the full models estimate of the cycle time.

5.4 Recommendations for Future Research

The aggregation methodology is an initial attempt to add analytical techniques to the process of creating an abstract, or aggregate simulation model. Areas where this work can be extended include:

(1) Estimating multiple performances characteristics of the flow line system from the aggregate simulation model:

This research estimates the average cycle time of a part through the flow line using the aggregate simulation model. Other performance characteristics such as the utilization, the cycle time, and queue waiting time of the individual resources are not computed. But, the aggregate methodology can possibly be extended to obtain estimates of these values. By running the aggregate simulation mode, one obtains an estimate of not only the average cycle time of a part, but also of the average cycle time of the part on each of the aggregation resources. Using the fact that an aggregation resource represents a group of resources which are related to one another by their distribution weights, preliminary results indicate that the average utilization of each resource can be obtained by multiplying its aggregation resource service utilization by the resource's distribution weight. For example, (say) R_2 (resource two) has two servers and a distribution weight of .43. Its utilization can be estimated by multiplying the service utilization of AR_2 (an output obtained by running the aggregate simulation model) by .43. It also appears that possibly an aggregation resource's cycle time can be partitioned using these weights to estimate the cycle time of each of the original resources. It is hypothesized that this involves solving a set of linear equations, similar to how the distribution weights were found. Thus, apply a similar (recursive?) technique.

(2) Including procedures for estimating the departure/arrival variability for G/G/S resources:

Using a procedure similar to that of the single server system, a procedure can be incorporated for estimating the output variability of a queue such that its impact can be incorporated into the aggregation. As the multiple server test cases indicate, modeling the resources as M/G/S queues under the independence assumption provide close approximations when the average utilization is high. Another possible approach is to artificially set (or increase) an aggregate resource's utilization to a high so that the simulation estimate is accurate. Once the simulation results are obtained, possibly the results are scaled (or reduced) to adjusts for the artificial increase.

(3) Incorporate feedback, rework, and scrap into the flow line.

To incorporate feedback, rework, and scrap into the aggregation procedure, the variability of the output process must be estimable. Currently only the single server type of system meets this requirement. The method to incorporate scrap appears quite straightforward in that after a part is finished being worked on by a resource, it has a certain probability of leaving the system (being scrap) and a certain probability of continuing to the next resource. Obviously the mean time between arrivals will reduce to: (mean time between arrivals) x (1 - probability of scrap). Correspondingly, the arrival variability must also be adjusted. Rework and feedback create an additional demand on a resource, for it modifies the arrival distribution (affecting both the mean and arrival variability). It appears that this can be incorporated using a similar procedure as scrap, except that rather than have parts leave the system, they return to a previous point within it, thus adding to the arrival mean and variability. These changes can possible lead to Assertion

4.1 of Section 4.3.3 to no longer be valid, in that coefficient of variation of the arrival time may be larger than one.

(4) Model resources with limited resource queue capacity:

The impact of limiting a queue's capacity involves estimating its effect on the upstream resources (prior resources in the flow line). The specific issue that needs to be addressed is: what happens when a queue is at capacity? Obviously, if no balking is allowed, when the system becomes blocked such that upstream resources can no longer send their parts to the blocked resource, the upstream resource become blocked themselves. Depending upon the length of the delay, this has a domino effect in that all the upstream resources will eventually be blocked. Thus, the part arrival pattern is being severely impacted by the queue capacity. A measure of this impact must be computed/calculated so that the aggregation methodology can incorporate it.

(5) Develop a procedure for measuring the reduction in model run time:

To fully apply or recommend the aggregation methodology, a measure must be developed for summarizing the gains of using it to model a flow line system. Simple time estimated can be determined by timing the run time (time until steady state is reached) of the full simulation model versus the run time of an aggregate simulation model for a series of case studies. A more ambitious approach is to mathematically determine or describe these events.

REFERENCES

Altiok, T., 1982. "Approximate Analysis of Exponential Tandem Queues and Blocking." *European Journal of Operations Research*. 11: 390-398.

Altiok, T., 1989. "Approximate Analysis of Queues in Series with Phase-Type Service Time and Blocking." *Operations Research*. 37: 601-610.

Anderson, D. R., D. J. Sweeney, and T. A. Williams, 1988. *An Introduction to Management Science: Quantitative Approaches to Decision Making*. St. Paul, Minnesota: West Publishing Company.

Aneke, N. A. G. and A. S. Carrie, 1984. "A Comprehensive Flowline Classification Scheme." *International Journal of Production Research*, 22: 281-297.

Annino, J. S. and E. C. Russell, 1979. "The Ten Most Frequent Causes of Simulation Analysis Failure - And How to Avoid Them." *Simulation*. 60: 137-140.

Antonelli, C., R. Volz, and T. Mudge, 1986. "Hierarchical Decomposition and Simulation of Manufacturing Cells Using Ada." *Simulation*. 46: 141-152.

Arthur, J. L., 1989. "Random Number Generation and Variance Reduction Techniques." Corvallis, Oregon: ST 491S Class Notes, Department of Statistics, Oregon State University.

Asimov, I., 1988. *Prelude to Foundation*. New York: Doubleday.

Balci, O., 1989. "How to Assess the Acceptability and Credibility of Simulation Results." *SCS 1989 Winter Simulation Conference*, pp. 62-71.

Balci, O. and R. E. Nance, 1987. "Simulation Support: Prototyping the Automation-Based Paradigm." *SCS 1987 Winter Simulation Conference*, pp. 495-502.

Banks, J. and J. S. Carson, 1984. *Discrete Event Simulation*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Boxma, O. J., J. W. Cohen, and N. Huffles, 1979. "Approximations for the Mean Waiting Time in an M/G/S Queueing System." *Operations Research*. 27: 1115-1127.

Brandwajin, A. and Y. L. Jow, 1988. "An Approximation Method for Tandem Queues With Blocking." *Operations Research*. 36: 73-83.

Bratley, P., B. L. Fox, and L. E. Schrage, 1983. *A Guide to Simulation*. New York: Springer-Verlag.

Budnick, F. S., R. Mojena, and T. E. Vollmann, 1977. *Principles of Operations Research for Management*. Homewood, Illinois: Richard D. Irwin, Inc.

Burke, P. J., 1956. "The Output of a Queueing System." *Operations Research*. 4: 699-704.

Carmichael, D. G., 1987. *Engineering Queues in Construction and Mining*. West Sussex, England: Ellis Horwood Limited.

Carson, J. S., 1986. "Convincing Users of Model's Validity Is Challenging Aspect of Modeler's Job." *Industrial Engineer*. 18: 74-85.

Cheng, T. C. E., 1990. "Analysis of Material Flow in a Job Shop with Assembly Operations." *International Journal of Production Research*. 28: 1369-1383.

Daley, D. J. and T. Rolski, 1992. "Light Traffic Approximations in Many-Server Queues." *Advanced Applied Probability*. 24: 202-218.

Derrick, E. J., O. Balci, and R. Nance, 1989. "A Comparison of Selected Conceptual Frameworks for Simulation Modeling." *SCS 1989 Winter Simulation Conference*, pp. 711-718.

Dietrich, B. L., 1991. "A Taxonomy of Discrete Manufacturing Systems." *Operations Research*. 39: 886-902.

Dietz, M., 1992. "Outline Of A Successful Simulation Project." *Simulation*. 24: 50-53.

Eklundh, B., 1981. "On the Accuracy of Simulation Programs." *SCS 1981 Summer Simulation Conference*, pp. 526-531.

Elmaghraby, S. E., 1968. "The Role of Modeling in I.E. Design." *Journal of Industrial Engineering*. XIX: 120-125

Emshoff, J. R. and R. L. Sisson, 1970. *Design and Use of Computer Simulation Models*. New York: Macmillan Publishing Company.

Erlang, A. K., 1909. "The Theory of probabilities and Phone Conversations." *Nyt. Tiddskrift Matematik B.* 20: 33-39.

Fishman, G. S., 1968. *Principles of Discrete Event Simulation*. New York: John Wiley and Sons.

Fossett, C. A., D. Harrison, and H. Weintrob, 1991. "An Assessment Procedure for Simulation Models: A Case Study." *Operations Research*. 39: 710-793.

Gershwin, S. B., 1987. "An Efficient Decomposition Method for the Approximate Evaluation of Tandem Queues with Finite Storage Space and Blocking." *Operations Research*, 35: 291-305.

Gershwin, S. B. 1989. "An Efficient Decomposition Algorithm for Unreliable Tandem Queueing Systems with Finite Buffers." H. G. Perros and T. Altiok, ed., *Queueing Networks with Blocking.* North-Holland, pp. 127-146.

Giffin, W. C., 1978. *Queueing: Basic Theory and Applications*. Columbus, Ohio: Grid, Inc.

Gnedenko, B. V. and I. N. Kovalenko, 1989. *Introduction to Queueing Theory*. Boston: Birkhauser.

Gordon, G., 1969. *System Simulation*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Graves, S. C., 1986. "A Tactical Planning Model for a Job Shop." *Operations Research*. 34: 522-533.

Groover, M. P., 1980. *Automation, Production Systems, and Computer-Aided Manufacturing*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Gross, D. and C. M. Harris, 1974. *Fundamentals of Queueing Theory*. New York: John Wiley and Sons.

Gun, L. and A. M. Makowski 1989. "An Approximation Method for General Tandem Queueing Systems Subject to Blocking." H. G. Perros and T. Altiok, ed., *Queueing Networks with Blocking.* North-Holland, pp. 147-174.

Hendricks, K. B., 1992. "The Output Processes of Serial Production Lines of Exponential Machines With Finite Queues." *Operations Research*. 40: 1139-1147.

Hillier, F. S. and R. W. Boling, 1966. "The Effect of Some Design Factors on the Efficiency of Production Lines with Variable Operation Times." *Journal of Industrial Engineering*. 17: 651-658.

Hillier, F. S. and R. W. Boling, 1967. "Finite Queues in Series with Exponential or Erlang Service Times - A Numerical Approach." *Operations Research*. 15: 286-303.

Hillier, F. S. and G. J. Lieberman, 1986. *Introduction to Operations Research* Oakland, California: Holden-Day, Inc.

Hokstad, P., 1978. "Approximations for the M/G/S Queue." *Operations Research*. 26: 510-523.

Hooper, J. W., 1986. "Activity Scanning and the Three-Phase Approach." *Simulation*. 47: 210-211.

Hunt, G. C., 1956. "Sequential Arrays of Waiting Lines." *Operations Research*. 4: 674-683.

Jacoby, S. L. S. and J. S. Kowalik, 1980. *Mathematical Modeling with Computers*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Kimura, T., 1986. "A Two-Moment Approximation for the Mean Waiting Time in GI/G/S Queues." *Management Science.* 32: 751-763.

Kimura, T., 1991. "Approximating the Mean Waiting Time in the GI/G/S Queue." *Journal of Operational Research Society*. 42: 959-970.

Kingman, J. F. C., 1962. "On Queues in Heavy Traffic." *Journal of the Royal Statistical Society, Series B*. 24: 383-392.

Kleinrock, L., 1976. *Queueing Systems: Volume II: Computer Applications*. New York: John Wiley and Sons.

Konig, D. and V. Schmidt, 1984. "Relationship Between Time/Customer Stationary Characteristics of Tandem Queues Attended by a Single Server." *Journal of the Operations Research Society of Japan*. 27: 191-204.

Kramer, W. and M. Langenbach-Belz, 1976. "Approximate Formulae for the Delay in the Queueing system GI/G/S." *8th International Teletraffic Congress*, pp. 235-248.

Kronmal, R. A. and A. V. Peterson, 1979. "On the Alias Method for Generating Random Variables From a Discrete Distribution." *The American Statistician*. 33: 214-218.

Ku, P.-S. and S.-C. Niu, 1986. "On Johnson's Two-Machine Flow Shop with Random Processing Times." *Operations Research*. 34: 130-36.

Lane, M. S., A. H. Mansour, and J. L. Harper, 1993. "Operations Research Techniques: A Longitudinal Update 1973-1988." *Interfaces*. 23: 63-68.

Law, A. M., 1986. "Introduction to Simulation: A Powerful Tool for Analyzing Complex Manufacturing Systems." *Industrial Engineer*. 18: 46-63.

Law, A. M. and W. D. Kelton, 1982. *Simulation Modeling and Analysis*. New York: McGraw Hill, Inc.

Law, A. M. and W. D. Kelton, 1991. *Simulation Modeling and Analysis*. Second Edition. New York: McGraw Hill.

Law, A. M. and M. G. McComas, 1986. “Pitfalls in the Simulation of Manufacturing Systems.” *SCS 1986 Winter Simulation Conference*, pp. 539-542.

Lee, A. M., 1966. *Applied Queueing Theory*. New York: Macmillan Publishing Company..

Lee, S. M., 1988. *Introduction to Management Science*. Chicago: The Dryden Press.

Lee, Y.-J. and P. Zipkin, 1992. “Tandem Queues With Planned Inventories.” *Operations Research*. 40: 936-947.

Lehmer, D. H., 1951. “Mathematical Models in Large-Scale Computing Units.” *Ann. Comp. Lab., Harvard University*. 26: 58-66.

Little, J. D. C., 1961. “A Proof of the Queueing Theory Formula L = λ W.” *Operations Research*. 9: 383-387.

Maaloe, E., 1973. “Approximations Formulae for Estimation of Waiting-Time in Multiple-Channel Queueing System.” *Management Science*. 19: 703-710.

Mackulak, G. T., J. K. Cochran, and D. L. Shunk, 1987. “Generic System Simulation Development on the PC (Phase I-III).” Tempe, Arizona: Systems Simulation Laboratory, Arizona State University,

MacNair, E. A. and C. H. Sauer, 1985. *Elements of Practical Performance Modeling*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Maisel, H. and G. Gnugnoli, 1972. *Simulation of Discrete Stochastic Systems*. Chicago: Science Research Associates, Inc.

Marchal, W. G., 1978. “Some Simpler Bounds on the Mean Queueing Time.” *Operations Research*. 26: 1083-1088.

Marsaglia, G., 1968. “Random Numbers Fall Mainly in the Planes.” *Proceedings of the National Academy of Science*, 60: pp. 25-28.

Marshall, K. T., 1968. “Some Inequalities in Queuing.” *Operations Research*. 16: 651-668.

McCormick, S. T., M. L. Pinedo, S. Shenker, and B. Wolf, 1989. “Sequencing in an Assembly Line with Blocking to Minimize Cycle Time.” *Operations Research*. 37: 925-935.

McHaney, R., 1991. *Computer Simulation - A Practical Perspective*. New York: Academic Press, Inc.

Medhi, J., 1991. *Stochastic Models in Queueing Theory*. New York: Academic Press, Inc.

Molloy, M. K., 1989. *Fundamentals of Performance Modeling*. New York: Macmillan Publishing Company.

Mori, M., 1975. "Some Bounds for Queues." *Journal of the Operations Research Society of Japan*. 18: 152-181.

Mott, J. and K. Tumay, 1992. "Developing a Strategy for Justifying Simulation." *Industrial Engineering*. 24: 38-42.

Murray, K. J. and S. V. Sheppard, 1987. "Automatic Model Synthesis: Using Automatic Programming and Expert Systems Techniques Toward Simulation Modeling." *SCS 1987 Winter Simulation Conference*, pp. 534-543.

Nance, R. E., 1983. "A Tutorial View of Simulation Model Development." *SCS 1983 Winter Simulation Conference*, pp. 325-331.

Naylor, T. H., 1979. *Simulation Models in Corporate Planning*. New York: Praeger Publishers.

Neelamkavil, F., 1987. *Computer Simulation and Modeling*. New York: John Wiley and Sons.

O'Keefe, R., 1986. "The Three-Phase Approach: A Comment on Strategy-Related Characteristics of Discrete-Event Languages and Models." *Simulation*. 47: 208-209.

Osborne, M. R. and R. O. Watts 1977. "Model Construction and Implementation." M. R. Osborne and R. O. Watts, ed., *Simulation and Modelling*, University of Queensland Press, pp. 3-29.

Overstreet, C. M. and R. E. Nance, 1985. "A Specification Language to Assist in Analysis of Discrete Event Simulation Models." *Communications of the ACM*. 28: 190-201.

Ozdemirel, N. E., 1990. "Generic Manufacturing Simulation Model Construction Based on Group Technology Classification of Models." Ph.D. Dissertation, Arizona State University.

Ozdemirel, N. E. and G. T. Mackulak, 1993. "A Generic Simulation Module Architecture Based on Clustering Group Technology Model Coding." *Simulation*. 60: 421-433.

Page, E., 1972. *Queueing Theory in OR*. London: The Bitterworth Group.

Pegden, C. D., R. E. Shannon, and R. P. Sadowski, 1990. *Introduction to Simulation Using SIMAN*. New York: McGraw-Hill, Inc.

Peterson, A. V. and R. A. Kronmal, 1982. "On Mixture Methods for the Computer Generation of Random Variables." *The American Statistician*. 36: 184-191.

Pinedo, M., 1982. "Minimizing the Expected Makespan in Stochastic Flow Shops." *Operations Research*. 30: 148-162.

Plane, D. R. and G. K. Kochenberger, 1972. *Operations Research for Managerial Decisions*. Homewood, Illinois: Richard D. Irwin, Inc.

Pollacia, L. F., 1989. "A Survey of Discrete Event Simulation and State-of-the-Art Discrete Event Languages." *Simulation Digest*. 18: 8-25.

Portier, F. J., 1987. "Implementing the Product Automaton Formalism." *SCS 1987 Winter Simulation Conference*, pp. 544-553.

Pritsker, A. A. B., 1986. *Introduction to Simulation and Slam II*. West Lafayette, Indiana: Systems Publishing Corporation.

Rao, B. M. and M. J. M. Posner, 1984. "On the Output Process of an M/M/1 Queue with Randomly Varying System Parameters." *Operations Research Letters*. 3: 191-197.

Ravindran, A., D. T. Phillips, and J. J. Solberg, 1987. *Operations Research: Principles and Practices*. New York: John Wiley and Sons.

Rogers, D. F., R. D. Plante, R. T. Wong, and J. R. Evans, 1991. "Aggregation and Disaggregation Techniques and Methodology in Optimization." *Operations Research*. 39: 553-582.

Ross, S., 1985. *Introduction to Probability Models*. New York: Academic Press.

Rozenblit, J. W., 1988. "Systems Theory Instrumented Simulation Modeling." *SCS 1988 Winter Simulation Conference*, pp. 282-286.

Rozenblit, J. W. and P. L. Jankowski, 1991. "An Integrated Framework for Knowledge-Based Modeling and Simulation of Natural Systems." *Simulation*. 57: 152-165.

Saaty, T. L., 1961. *Elements of Queueing Theory*. New York: McGraw Hill.

Sadowski, R. P., 1989. "The Simulation Process: Avoiding the Problems and Pitfalls." *SCS 1989 Winter Simulation Conference*, pp. 72-79.

Sargent, R. G., 1987. "On Overview of Verification and Validation of Simulation Models." *SCS 1987 Winter Simulation Conference*, pp. 33-39.

Shoemaker, S. 1978. "On Simulation." S. Shoemaker, ed., *Computer Networks and Simulation,* Elsevier Science Publishers (North-Holland), pp. 69-84.

Schruben, L., 1983. "Modeling Systems Using Discrete Event Simulation." *SCS 1983 Winter Simulation Conference*, pp. 101-107.

Schweitzer, P. J. and T. Altiok 1989. "Aggregate Modelling of Tandem Queues Without Intermediate Buffers." H. G. Perros and T. Altiok, ed., *Queueing Networks with Blocking.* North-Holland, pp. 47-72.

Seelen, L. P. and H. C. Tijms, 1984. "Approximations for the Conditional Waiting Times in the GI/G/C Queue." *Operations Research Letters*. 3: 183-190.

Shalmon, M. and M. A. Kaplan, 1984. "A Tandem Network of Queues with Deterministic Service and Intermediate Arrivals." *Operations Research*. 32: 753-773.

Shannon, R. E., 1975. *Systems Simulation the Art and Science*. Englewood Cliffs, New Jersey: Prentice-Hall.

Shannon, R. E., R. Mayer, and H. H. Adelsberger, 1985. "Expert Systems and Simulation." *Simulation*. 44: 275-284.

Shanthikumar, J. G., 1983. "Bounds and an Approximation for Single Server Queues." *Journal of the Operations Research Society of Japan*. 26: 118-133.

Stoyan, D., 1976. "Approximations for M/G/s queues." *Mathematics of Operations Research*. 7: 587-594.

Suresh, S. and W. Whitt, 1990. "Arranging Queues in Series: A Simulation Experiment." *Management Science*. 36: 1080-1091.

Suzuki, T. and Y. Yoshida, 1970. "Inequalities for Many-Server Queue and Other Queues." *Journal of the Operations Research Society of Japan*. 13: 59-77.

Takahashi, Y., 1977. "An Approximation Formula for the Mean Waiting Time of an M/G/C Queue." *Journal of the Operations Research Society of Japan*. 20: 150-163.

Takahashi, Y. 1989. “Aggregate Approximation for Acyclic Queueing Networks with Communication Blocking.” H. G. Perros and T. Altiok, ed., *Queueing Networks with Blocking.* North-Holland, pp. 33-46.

Takahashi, Y., H. Miyahara, and T. Hasegawa, 1980. “An Approximation Method for Open Restricted Queuing Networks.” *Operations Research*. 28: 594-602.

Tausworthe, R. C., 1965. “Random Numbers Generated by Linear Recurrence Modulo Two.” *Math. Comput.* 19: 201-209.

Thesen, A. and L. Travis, 1988. “Introduction to Simulation.” *SCS 1988 Winter Simulation Conference*, pp. 7-14.

Tijms, H. C., M. H. V. Hoorn, and A. Federgruen, 1981. “Approximations for the Steady-State Probabilities in the M/G/C Queue.” *Advanced Applied Probability*. 13: 186-206.

Ulgen, O. M., “Proper Management Techniques Are Keys to A Successful Simulation Project.” *Industrial Engineering*. 23: 37-41.

Wagner, H. M., 1969. *Principles of Operations Research with Applications to Managerial Decisions*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Weiner, S. A., J. W. Grant, and P. E. Coffman, 1986. “Manufacturing Simulation in Practice: Suggestions for Establishing Credibility - An Extended Example.” *SCS 1986 Winter Simulation Conference*, pp. 543-544.

Whitt, W., 1983. “Approximating a Point Process by a Renewal Process: Two Basic Methods.” *Operations Research*. 31: 125-147.

Whitt, W., 1983. “Comparison Conjectures about the M/G/S Queue.” *Operations Research Letters*. 2: 203-209.

Whitt, W., 1984. “Departures from a Queue with Many Busy Servers.” *Mathematics of Operations Research*. 9: 534-544.

Wittrock, R. J., 1988. “An Adaptive Scheduling Algorithm For Flexible Flow Lines.” *Operations Research*. 36: 445-453.

Wolff, R. W., 1982. “Tandem Queues with Dependent Service Times in Light Traffic.” *Operations Research*. 30: 619-635.

Wolff, R. W., 1989. *Stochastic Modeling and the Theory of Queues*. Englewood Cliffs, New Jersey: Prentice Hall.

Zeigler, B., 1986. “Hierarchical Modular Modeling/Knowledge Representation.” *SCS 1986 Winter Simulation Conference*, pp. 129-137.

Zeigler, B. P., 1987. “Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Environment.” *Simulation*. 49: 219-230.

Zeigler, B. P. and T. Oren, 1986. “Multifaceted, Multiparidigm Modelling Perspectives: Tools for the 90’s.” *SCS 1986 Winter Simulation Conference*, pp. 708-712.

APPENDIX A

SLAM SIMULATION MODELS FOR MPD EXAMPLE

```
;****  Full Model of MPD Manufacturing System  - all resources explicitly modeled

GEN, Savory, MPD Full,10/15/93,30,,,,,,72;
LIMITS,6,1,2000;
;
NETWORK:
        CREATE,EXPON(100),,1;
;
F1      COLCT,BET,TIME BET. ARR 1;
        QUEUE(1);
        ACT(1)/1,UNFRM(75,85);
;
F2  COLCT,BET,TIME BET. ARR 2;
        QUEUE(2);
        ACT(2)/2,RNORM(130,15);
;
F3      COLCT,BET,TIME BET. ARR 3;
        QUEUE(3);
        ACT(2)/3,TRIAG(120,150,180);
;
F4      COLCT,BET,TIME BET. ARR 4;
        QUEUE(4);
        ACT(4)/4,RNORM(320,25);
;
F5      COLCT,BET,TIME BET. ARR 5;
        QUEUE(5);
        ACT(1)/5,TRIAG(32,43,60);
;
F6      COLCT,BET,TIME BET. ARR 6;
        QUEUE(6);
        ACT(1)/6,UNFRM(64,80);
;
BYE     COLCT,BET,TIME BET. ARR BYE;
        COLCT,INT(1),TIME IN SYSTEM;
        TERM,250000;
        END;
FIN;
```

```
;****  Aggregate Model of MPD Manufacturing System  - resources modeled as
aggregates

GEN, Savory, MPD Agg,10/15/93,30,,,,,72;
LIMITS,16,6,1000;
NETWORK;
        CREATE,EXPON(100),,1;
;
        COLCT,BET, TM BET. ARRIVAL;
        ASSIGN, II=II+1;
        ASSIGN, ATRIB(6)=II;
;
        GOON,3;
            ACT,,,AR1;
            ACT,,,AR2;
            ACT,,,AR4;
;
;=============================================================
;
AR1         COLCT,BET,TIME BET. ARR 1;
        GOON,1;
            ACT,,.465724,A11;
            ACT,,.186597,A12;
            ACT,,.347679,A13;
A11     ASSIGN, ATRIB(3)=UNFRM(75,85);
            ACT,,,D1;
A12     ASSIGN, ATRIB(3)=TRIAG(32,43,60);
            ACT,,,D1;
A13     ASSIGN, ATRIB(3)=UNFRM(64,80);
            ACT,,,D1;
;
D1 Queue(1);
            ACT(1)/1,ATRIB(3);
;
        ASSIGN,ATRIB(2)=TNOW-ATRIB(1)-ATRIB(3);
        ASSIGN,ATRIB(4)=ATRIB(2)+ATRIB(3);
        COLCT,ATRIB(3),AR1 SERVICE TM;
        COLCT,ATRIB(4),AR1 CYCLE TM;
        ASSIGN,ATRIB(5)=ATRIB(4)*3;
        COLCT,ATRIB(5),AR1 TOTAL CYCLE;
            ACT,,,GO1;
;
;=============================================================
;
AR2     COLCT,BET,TIME BET. ARR 2;
```

```
        GOON,1;
            ACT,,.431361,A21;
            ACT,,.568639,A22;
A21     ASSIGN, ATRIB(3)=RNORM(130,15);
            ACT,,,D2;
A22     ASSIGN, ATRIB(3)=TRIAG(120,150,180);
            ACT,,,D2;
;
D2 Queue(2);
            ACT(2)/2,ATRIB(3);
;
        ASSIGN,ATRIB(2)=TNOW-ATRIB(1)-ATRIB(3);
        ASSIGN,ATRIB(4)=ATRIB(2)+ATRIB(3);
        COLCT,ATRIB(3),AR2 SERVICE TM;
        COLCT,ATRIB(4),AR2 CYCLE TM;
        ASSIGN,ATRIB(5)=ATRIB(4)*2;
        COLCT,ATRIB(5),AR2 TOTAL CYCLE;
            ACT,,,GO2;
;
;============================================================
;
AR4     COLCT,BET,TIME BET. ARR 4;
        GOON,1;
            ACT,,1,A41;
A41     ASSIGN, ATRIB(3)=RNORM(320,25);
            ACT,,,D4;
;
D4 Queue(4);
            ACT(4)/4,ATRIB(3);
;
        ASSIGN,ATRIB(2)=TNOW-ATRIB(1)-ATRIB(3);
        ASSIGN,ATRIB(4)=ATRIB(2)+ATRIB(3);
        COLCT,ATRIB(3),AR4 SERVICE TM;
        COLCT,ATRIB(4),AR4 CYCLE TM;
        ASSIGN,ATRIB(5)=ATRIB(4)*1;
        COLCT,ATRIB(5),AR4 TOTAL CYCLE;
            ACT,,,GO4;
;
;============================================================;
GO1     QUEUE(9),,,,MTCH;
GO2     QUEUE(10),,,,MTCH;
GO3     QUEUE(11),,,,MTCH;
GO4     QUEUE(12),,,,MTCH;
GO5     QUEUE(13),,,,MTCH;
GO6     QUEUE(14),,,,MTCH;
```

```
GO7     QUEUE(15),,,,MTCH;
GO8     QUEUE(16),,,,MTCH;
;
MTCH    MATCH,6,GO1/AA,GO2/AA,GO4/AA;
AA      ACCUM,3,3,SUM;
;
BYE     COLCT,BET,TIME BET. ARR BYE;
        COLCT,ATRIB(5),TIME IN SYSTEM;
        TERM,250000;
        END;
FIN;
```

APPENDIX B

MATHEMATICA CODE FOR AGGREGATION PROGRAM

Aggregation[]:=

```
Module[{},
      typeRun = Input["Enter Parameters or Randomly Generate?\n
                  [ 0  = Enter   1 = Generate ]  "];
      (* Test whether generate a flow line or have user enter it *)
      If[(typeRun == 0),
          UserMain[],
          GenerateMain[]
      ]
]
```

APrint[]:=

```
Module[{},

(* Print out the Aggregation Description *)
Print["  "];
Print["Aggregate Flow Line Description..."];

(* Find service means for each aggregation block *)
For [i=1, i <= maxServers, i++,

If[(numServer[i] > 0),

Print[" "];
Print["Aggregate Resource #",i];
Print["  Arrival Mean is ",TBA];
Print["  Arrival Rate is ",N[L]];
Print["  Total AR Cycle Time is ",TWAR[i]];
Print["  Average AR Cycle Time is ",AAR[i]];
Print["  Est. Service Time SQCOV is ",estSQCV[i]];
Print["  Est. AR Service Mean is ",MAR[i]];
Print["  Est. AR Service Rate is ",N[1/MAR[i]]];
Print["  Distribution Weights for AR #",i];
For[j=1,j<=numServer[i],j++,
       For[k=1,k<=numServer[i],k++,
          If[(where[i,k] == j),
             Print["    Weights of Resource #",rNum[i,k]," is ",DWT[i,k]];
         ]
       ]
]
]
]
]
```

DistWeight1[i_, start_, total_, numAR_, meanAR_, sumAR_]:=

```
Module [{temp1,temp2,temp3,temp4},


    (* One resource is aggregated *)
    If[(numAR == 1),
        DWT[i,start] = 1;
    ]

    (* Two resources are aggregated, solve for weights *)
    If[(numAR == 2),
        Clear[xX, yY];
        If[(tAVG[i,start] != tAVG[i,(start+1)]),
        Unprotect[temp1];
        temp1=Solve[{((xX*tAVG[i,start])+(yY*tAVG[i,(start+1)]))==meanAR,xX+yY ==
                sumAR},{xX,yY}];
        DWT[i,start]=Part[(xX/.temp1), 1];
        DWT[i,(start+1)]=Part[(yY/.temp1), 1];

         If[(tAVG[i,start] == tAVG[i,(start+1)]),
           DWT[i,start] = .500;
           DWT[i,(start+1)] = .500;
         ]
    ]

    (* More than two resoruces aggregated, aggregate fist two and recurse *)
    If[(numAR > 2),
        (* Determine new aggregate resource total and average cycle time *)

        rTWT = (tWT[i,start]+tWT[i,(start+1)]);
        rAVG = rTWT/2;

        (* Determine estimated squared COV *)
        rSQCV = (((tWT[i,start]/rTWT)*tSQCV[i,start])
              +((tWT[i,(start+1)]/rTWT)*tSQCV[i,(start+1)]));

        (* Determing estimated mean for new aggregate resource *)
        MGSServiceMean[i, rSQCV, rAVG];

        tWT[i,(total+1)] = rAVG;
        tSQCV[i,(total+1)] = rSQCV;
        tAVG[i,(total+1)] = meanVal;
        (*tAVG[i,(total+1)] = Input["Enter meanVal"];*)

        DistWeight1[i, (start+2), (total+1),(numAR-1),meanAR,sumAR];
        Unprotect[temp1];
        temp1 = meanAR - (meanAR - (tAVG[i,(total+1)]*DWT[i,(total+1)]));
        Unprotect[temp2];
        temp2= DWT[i,(total+1)];
        If[((start > 0)&&(DWT[i,(total+1)])>0),
           DistWeight1[i, start, total, 2,temp1,temp2]];
    ];
];
```

DistWeight2[i_, start_, numAR_]:=

```
Module [{},

    (* One resource is aggregated *)
    If[(numAR == 1),
        DWT[i,start] = 1;
    ]

    (* Two resources are aggregated, solve for weights *)
    If[(numAR == 2),
        Clear[xX, yY];
        If[(tAVG[i,start] != tAVG[i,(start+1)]),
        temp1=Solve[{((xX*tAVG[i,start])+(yY*tAVG[i,(start+1)]))==meanAR,xX+yY ==
                sumAR},{xX,yY}];
        DWT[i,start]=Part[(xX/.temp1), 1];
        DWT[i,(start+1)]=Part[(yY/.temp1), 1];
         ]

         If[(tAVG[i,start] == tAVG[i,(start+1)]),
           DWT[i,start] = .500;
           DWT[i,(start+1)] = .500;
         ]

        Unprotect[meanAR];
        meanAR = meanAR - (tAVG[i,(start+1)]*DWT[i,(start+1)]);
        Unprotect[sumAR];
        sumAR= sumAR - DWT[i,(start+1)];

        tWT[i,start] = WT[i,start];
        tSQCV[i,start] = SQCV[i,start];
        tAVG[i,start] = AVG[i,start];

        If[((start > 0)&&(DWT[i,(start+1)]>0)&&(DWT[i,start]>0)),
              DistWeight2[i, (start - 1), numAR]];
    ]

    (* More than two resoruces aggregated, aggregate fist two and recurse *)
    If[(numAR > 2),

        (* Determine new aggregate resource total and average cycle time *)

        rTWT = (tWT[i,start]+tWT[i,(start+1)]);
        rAVG = rTWT/2;
        (* Determine estimated squared COV *)
        rSQCV = (((tWT[i,start]/rTWT)*tSQCV[i,start])
              +((tWT[i,(start+1)]/rTWT)*tSQCV[i,(start+1)]));
        (* Determing estimated mean for new aggregate resource *)
        MGSServiceMean[i, rSQCV, rAVG];

        tWT[i,(start+1)] = rAVG;
        tSQCV[i,(start+1)] = rSQCV;
        tAVG[i,(start+1)] = meanVal;
        DistWeight2[i, (start+1), (numAR-1)];
    ]
```

]

enterGenServer[]:=

```
Module[{},

    (* Check if single server queue *)
    If[(singleCheck == 0),
        servers = Random[DiscreteUniformDistribution[8]],
        servers = 1
    ]

    (* Keep track of largest number of servers in the flow line *)
    If[(servers > maxServers), maxServers = servers];

    (* Count the number of the each type of servers *)
    numServer[servers]=numServer[servers]+1;
    numSame=numServer[servers];

    (* Store the row and column of the matrix where stored *)
    origRow[i] = servers;
    origCol[i] = numSame;
]
```

enterServer[]:=

```
Module[{},

    (* Check if single server queue *)
    If[(singleCheck == 0),
        servers=Input["Number of Parallel Servers for the Resource?"],
        servers=1
    ]

    (* Keep track of largest number of servers in the flow line *)
    If[(servers > maxServers), maxServers = servers];

    (* Count the number of the each type of servers *)
    numServer[servers]=numServer[servers]+1;
    numSame=numServer[servers];

    (* Store the row and column of the matrix where stored *)
    origRow[i] = servers;
    origCol[i] = numSame;
]
```

FPrint[]:=

```
Module[{},

(* Print out flow line description *)
Print["Flow Line Description......."];

For[i=1,i<=numQue,i++,

(* Create temporary variables - find position *)
aA = origRow[i];
bB = origCol[i];

Print[" "];
Print["Resource #",i," of the Flow Line"];
Print["  Arrival Mean is ",TBA];
Print["  Arrival Rate is ",N[L]];
Print["  Arrival Variability is ",VA[aA,bB]];
Print["  Arrival COV is ",N[Sqrt[VA[aA,bB]]/TBA]];
Print["  Arrival SQCOV is ",N[VA[aA,bB]]/(TBA^2)];
Print["  Number of Parallel Resources is ",aA];
Print["  ",dD[aA,bB]];

Print["  Service Mean is ",AVG[aA,bB]];
Print["  Service Variability is ",VAR[aA,bB]];
Print["  Service SD is ",Sqrt[VAR[aA,bB]]];
Print["  Service Time is COV is ",CV[aA,bB]];
Print["  Service Time SQCOV is ",SQCV[aA,bB]];
Print["  Resource Utilization is ", (L*AVG[aA,bB])/aA];
Print["  Est. Queue Waiting Time is ",WTQ[aA,bB]];
Print["  Est. Resource Cycle Time is ",WT[aA,bB]];

]
]
```

GenerateMain[]:=

```
Module[{},

        (* Read in what type of case to generate *)
      typeCase = Input["Generate: 1 = Exponential Test Case\n
                                  2 = Single Server Test Case\n
                                  3 = Multiple Server Test Case"];

       numTest = Input["Number of Test Cases to Generate?"];


        numQue = Random[DiscreteUniformDistribution[16]]+4;

          Print[" "];
          If[(typeCase == 1),  Print["Exponential Test Case #", mpd," (",numQue," Queues) of
    ",numTest]];
        If[(typeCase == 2),  Print["Single Server Test Case #",mpd,"(",numQue," Queues) of ",numTest]];
        If[(typeCase == 3),  Print["Mulitple Server #",mpd,"(",numQue," Queues) of ",numTest]];

        (* Set flag as to whether this is a single case - thus require compute variance *)
        If[(typeCase == 2),
          singleCheck = 1,
          singleCheck = 0;
        ]

        (* Generate actual data *)
        Step1Generate[];

        If[(singleCheck == 0),
            Step2[],
            Step2Single[];
        ]

        Step3[];
        FPrint[];
        Step4[];
        APrint[];
      ]
]
```

GG1CycleTime[j_,wqPrevious_]:=

```
Module[{},

    (* Compute rho for previous resource *)
    r = L*AVG[1,(j-1)];

    (* Compute the variability of the arrival process *)
    VA[1,j]=arrivalVar+(2*VAR[1,(j-1)])-((2/L)*(1-r)*wqPrevious);

    (* Squared coefficient of variaton of arrival process *)
    sqcvA = VA[1,j] / (TBA^2);

    (* Compute rho for current resource *)
    r = L*AVG[1,j];

    (* compute the estimated waiting time *)
    wqMM1=r/((1/AVG[1,j])-L);

    g=Exp[(-2*(1-r)*((1-sqcvA)^2)) / (3*r*(sqcvA+SQCV[1,j]))];

    (* Queue waiting time *)
    wqGG1=((sqcvA+SQCV[1,j])/2)*g*wqMM1;

    (* Cycle time *)
    wGG1 = wqGG1 + AVG[1,j];

    arrivalVar = VA[1,j];
    Clear[wqMM1,g,sqcvA];

]
```

HighLow[i_]:=

```
Module[{Temp1, Temp2,Temp3, Temp4, Temp5},

      (* Perform a selection sort and swap values - original values are lost *)
       For[j=2, j < numServer[i],,

          Temp1=AVG[i,j];
          Temp2=WT[i,j];
          Temp3=SQCV[i,j];
          Temp4=rNum[i,j];
          Temp5=where[i,j];

          AVG[i,j]=AVG[i,numServer[i]];
          WT[i,j]=WT[i,numServer[i]];
          SQCV[i,j]=SQCV[i,numServer[i]];
          rNum[i,j]=rNum[i,numServer[i]];
          where[i,j]=where[i,numServer[i]];

        For[k=numServer[i], k >= (j+1), k--,
            AVG[i,k]=AVG[i,(k-1)];
              WT[i,k]=WT[i,(k-1)];
              SQCV[i,k]=SQCV[i,(k-1)];
              rNum[i,k]=rNum[i,(k-1)];
              where[i,k]=where[i,(k-1)];
          ];

          AVG[i,(j+1)] = Temp1;
          WT[i,(j+1)] = Temp2;
          SQCV[i,(j+1)] = Temp3;
          rNum[i,(j+1)] = Temp4;
          where[i,(j+1)] = Temp5;

          j = j + 2;
      ];
];
```

MGSCycleTime[i_,j_]:=

```
Module [{},

    LM = L * AVG[i,j];

    (* Compute rho *)
    r = LM / i;

    (* Probability in state zero *)
    pO = (Sum[(LM^n)/Factorial[n],{n,0,(i-1)}])
          + ((LM^i)/(Factorial[i]*(1-r)));
    pO = 1 / pO;

    (* Erlangs loss *)
    cC = ((LM^i) / (Factorial[i]*(1-r)))*pO;

    (* Queue waiting time *)
    wqMGS = ((1+SQCV[i,j])/(2*L*(1-r)))*r*cC;

    (* Cycle time *)
    wMGS = wqMGS + AVG[i,j];

    Clear[LM,r,pO,cC];
]
```

MGSServiceMean[i_,estSQCV_,cycle_]:=

```
Module [{},

    LM = L * zzZZ;

    (* Compute rho *)
    r = LM / i;

    (* Probability in state zero *)
    pO = (Sum[(LM^n)/Factorial[n],{n,0,(i-1)}])
          + ((LM^i)/(Factorial[i]*(1-r)));
    pO = 1 / pO;

    (* Erlangs loss *)
    cC = ((LM^i) / (Factorial[i]*(1-r)))*pO;

    (* Queue waiting time *)
    wqMGS = ((1+estSQCV)/(2*L*(1-r)))*r*cC;

    (* Cycle time *)
    wMGS = wqMGS + zzZZ;

    temp1 = Solve[wMGS == cycle, zzZZ];

    (* Remove "zzZZ ->" arrow from solutions, thus all numbers now *)
    temp2 = zzZZ/.temp1;
    numList = Length[temp2];

    (* Pick among the solutions for the one which satisfies rho requirements *)
    For[j=1, j<=numList, j++,
          (* Read the real part of number j from the list *)
        temp3 = Re[Part[temp2,j]];

        (* Compute rho with this number *)
        temp4 = ((L * temp3) / i);

        (* Check if it satisfies rho *)
        If[((temp4 > 0)&&(temp4 < 1)),
            meanVal = temp3;
        ]
    ]
    Clear[temp1, temp2, temp3, temp4, zzZZ];
]
```

Step1[]:=

```
Module[{},

    (* Initialize default values on number of each type of server *)
    (* Note limit of 25 servers -> 25 Aggregation resources *)
    maxServers = 0;
    For[i=1, i<=25, i++,
        numServer[i] = 0;
    ]

    Unprotect[numQue];
    numQue=Input["Number of Resources in the Flow Line?"];
    singleCheck=Input["Does the Flow Line Have only Single Capacity Servers? ( 0 = No  1 = Yes )"];

    (* read in the time between arrivals *)
    TBA = Input["Time Between Part Arrivals?"];
    L = 1/TBA;

    (* Read in, compute, and store information for each resource *)
    For[i=1, i <= numQue, i++,

    (* Read in the service distribution *)
    temp1= "Select the Service Distribution for Resource #";
    temp2= ToString[i];
    temp3 = StringJoin[temp1,temp2,":  \n\n
                          1 = Exponential    4  = Triangular\n
                          2 = Lognormal      5  = Uniform\n
                         3 = Normal"];
    choice=Input[temp3];

    idString= StringJoin["Resource #",temp2,":  "];


    (* Service Distribution is Exponential *)
    If[(choice == 1),
        aa=Input[StringJoin[idString,"Service Mean"]];
        enterServer[];

        AVG[servers,numSame]=N[aa];
        VAR[servers,numSame]=N[aa^2];
        CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
        SQCV[servers,numSame]=CV[servers,numSame]^2;

        (* store which distibution and the parameters *)
        dist1="Service Dist is Exponential (";
        dist2=ToString[aa];
        dist3=")";
        dD[servers,numSame] = StringJoin[dist1,dist2, dist3];
    ]

    (* Service Distribution is Lognormal *)
    If[(choice == 2),
        aa=Input[StringJoin[idString,"Mu"]];
        bb=Input[StringJoin[idString,"Standard Deviation"]];
```

```
        bb=bb^2;
        enterServer[];

        AVG[servers,numSame]=N[aa];
        VAR[servers,numSame]=N[bb];
        CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
        SQCV[servers,numSame]=CV[servers,numSame]^2;

        (* store which distibution and the parameters *)
        c=", ";
        dist1="Service Dist is Lognormal (";
        dist2=ToString[aa];
        dist3=ToString[Sqrt[bb]];
        dist4=")";
        dD[servers,numSame] = StringJoin[dist1,dist2, c, dist3,dist4];
]

(* Service Distribution is Normal *)
If[(choice == 3),
        aa=Input[StringJoin[idString,"Mu"]];
        bb=Input[StringJoin[idString,"Standard Deviation"]];
        bb=bb^2;
        enterServer[];

        AVG[servers,numSame]=N[aa];
        VAR[servers,numSame]=N[bb];
        CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
        SQCV[servers,numSame]=CV[servers,numSame]^2;

        (* store which distibution and the parameters *)
        c=", ";
        dist1="Service Dist is Normal (";
        dist2=ToString[aa];
        dist3=ToString[Sqrt[bb]];
        dist4=")";
        dD[servers,numSame] = StringJoin[dist1,dist2, c, dist3,dist4];
]

(* Service Distribution is Triangular *)
If[(choice == 4),
        aa=Input[StringJoin[idString,"Minimium"]];
        bb=Input[StringJoin[idString,"Mode"]];
        cc=Input[StringJoin[idString,"Maximium"]];
        enterServer[];

        AVG[servers,numSame]=N[(aa+bb+cc)/3];
        VAR[servers,numSame]=N[(aa^2+bb^2+cc^2-(aa*bb)-(aa*cc)-(bb*cc))/18];
        CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
        SQCV[servers,numSame]=CV[servers,numSame]^2;

        (* store which distibution and the parameters *)
        c=", ";
        dist1="Service Dist is Triag (";
        dist2=ToString[aa];
```

```
        dist3=ToString[bb];
        dist4=ToString[cc];
        dist5=")";
        dD[servers,numSame] = StringJoin[dist1,dist2,c,dist3,c,dist4,dist5];
    ]

    (* Service Distribution is Uniform *)
    If[(choice == 5),
        aa=Input[StringJoin[idString,"Minimium"]];
        bb=Input[StringJoin[idString,"Maximium"]];
        enterServer[];

        AVG[servers,numSame]=N[(aa+bb)/2];
        VAR[servers,numSame]=N[((bb-aa)^2)/12];
        CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
        SQCV[servers,numSame]=CV[servers,numSame]^2;

        (* store which distibution and the parameters *)
        c=", ";
        dist1="Service Dist is Uniform (";
        dist2=ToString[aa];
        dist3=ToString[bb];
        dist4=")";
        dD[servers,numSame] = StringJoin[dist1,dist2,c, dist3,dist4];
    ]

    (* Store which resource this is *)
    Unprotect[rNum[servers,numSame]];
    rNum[servers,numSame] = i;

    Clear[choice,dist1,dist2,dist3,dist4,dist5,aa,bb,cc];
    Clear[temp1, temp2, temp3, idString];

    (* Compute rho *)
    temp1=(L*AVG[servers,numSame])/servers;

    (* Test whether rho is in range *)
    If[((temp1 >= 1)||(temp1 < 0)),
       Print[" "];
       Print["<<<<<    ERROR: Resource #",i," >>>>> "];
       Print["<<<<<     Rho >=1 or Rho < 1   >>>>>>"];
       Abort[];
    ]
    ]
]
```

Step1Generate[]:=

```
Module[{},

    (* Initialize default values on number of each type of server *)
    (* Note limit of 25 servers -> 25 Aggregation resources *)
    maxServers = 0;
    For[i=1, i<=25, i++,
        numServer[i] = 0;
    ]

    (* Arrival rate is set to a constant of 100 *)
    Unprotect[TBA];
    TBA = 100;
    L = 1/TBA;

    (* Read in, compute, and store information for each resource *)
    For[i=1, i <= numQue, i++,

    (* Set Value of Rho *)
    rho = Random[UniformDistribution[.20,.80]];

    (* Select a service distribution *)
    If[(typeCase == 1),
        choice = 1,
        choice = Random[DiscreteUniformDistribution[5]]
    ]

    (* Service Distribution is Exponential *)
    If[(choice == 1),
        enterGenServer[];

        (* compute mean using rho *)
        aa=rho*servers*TBA;
        AVG[servers,numSame]=N[aa];
        VAR[servers,numSame]=N[aa^2];
        CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
        SQCV[servers,numSame]=CV[servers,numSame]^2;

        (* store which distibution and the parameters *)
        dist1="Service Dist is Exponential (";
        dist2=ToString[aa];
        dist3=")";
        dD[servers,numSame] = StringJoin[dist1,dist2, dist3];
    ]


    (* Service Distribution is Lognormal *)
    If[(choice == 2),
        enterGenServer[];

        (* compute mean using rho *)
        aa=N[rho*servers*TBA];

        (* compute sd using a generated coefficient of variation *)
```

```
        bb=(Random[UniformDistribution[.01,.10]])*aa;
        bb=N[bb^2];

        AVG[servers,numSame]=N[aa];
        VAR[servers,numSame]=N[bb];
        CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
        SQCV[servers,numSame]=CV[servers,numSame]^2;

        (* store which distibution and the parameters *)
        c=", ";
        dist1="Service Dist is Lognormal (";
        dist2=ToString[aa];
        dist3=ToString[Sqrt[bb]];
        dist4=")";
        dD[servers,numSame] = StringJoin[dist1,dist2, c, dist3,dist4];
]

(* Service Distribution is Normal *)
If[(choice == 3),
          enterGenServer[];

        (* compute mean using rho *)
        aa=rho*servers*TBA;

        (* compute sd using a generated coefficient of variation *)
        bb=(Random[UniformDistribution[.01,.10]])*aa;
        bb=bb^2;

        AVG[servers,numSame]=N[aa];
        VAR[servers,numSame]=N[bb];
        CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
        SQCV[servers,numSame]=CV[servers,numSame]^2;

        (* store which distibution and the parameters *)
        c=", ";
        dist1="Service Dist is Normal (";
        dist2=ToString[aa];
        dist3=ToString[Sqrt[bb]];
        dist4=")";
        dD[servers,numSame] = StringJoin[dist1,dist2, c, dist3,dist4];
]

(* Service Distribution is Triangular *)
If[(choice == 4),

        enterGenServer[];

        (* Compute mean using rho *)
        Unprotect[mean];
        mean=rho*servers*TBA;

        pass = 0;

        For[qq=1,pass == 0, qq++,
```

```
            (* Generate maximum *)
            cc = Random[UniformDistribution[0,TBA]];

        (* Generate mode *)
        bb = Random[UniformDistribution[0, cc]];

        (* Generate minimium *)
        aa = Random[UniformDistribution[0, bb]];

        (* Compute Mean value for this parameter set *)
        Unprotect[temp];
        temp = (aa+bb+cc)/3;

        (* Quit for if the mean of this parameter set is less than req/ given rho *)
        If[(temp <= mean)), pass = 1]
    ]

    Unprotect[AVG[servers,numSame]];
    AVG[servers,numSame]=N[(aa+bb+cc)/3];
    VAR[servers,numSame]=N[(aa^2+bb^2+cc^2-(aa*bb)-(aa*cc)-(bb*cc))/18];
    CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
    SQCV[servers,numSame]=CV[servers,numSame]^2;

    (* store which distibution and the parameters *)
    c=", ";
    dist1="Service Dist is Triag (";
    dist2=ToString[aa];
    dist3=ToString[bb];
    dist4=ToString[cc];
    dist5=")";
    dD[servers,numSame] = StringJoin[dist1,dist2,c,dist3,c,dist4,dist5];
]

(* Service Distribution is Uniform *)
If[(choice == 5),

    enterGenServer[];

    (* Compute mean using rho *)
    mean=N[rho*servers*TBA];

    (* Compute SD using a generated coefficient of variation *)
    sd=N[(Random[UniformDistribution[.01,.10]])*mean];

    (* Compute min and max ranging values from the mean and sd *)
    aa = N[mean - (Sqrt[3]*sd)];
    bb = N[mean + (Sqrt[3]*sd)];

    AVG[servers,numSame]=N[(aa+bb)/2];
    VAR[servers,numSame]=N[((bb-aa)^2)/12];
    CV[servers,numSame]=Sqrt[VAR[servers,numSame]]/AVG[servers,numSame];
    SQCV[servers,numSame]=CV[servers,numSame]^2;
```

```
        (* store which distibution and the parameters *)
        c=", ";
        dist1="Service Dist is Uniform (";
        dist2=ToString[aa];
        dist3=ToString[bb];
        dist4=")";
        dD[servers,numSame] = StringJoin[dist1,dist2,c, dist3,dist4];
    ]

    (* Store which resource this is *)
    Unprotect[rNum[servers,numSame]];
    rNum[servers,numSame] = i;

    Clear[choice,dist1,dist2,dist3,dist4,dist5,aa,bb,cc];
    Clear[temp1, temp2, temp3, idString];

    (* Compute rho *)
    Unprotect[temp1];
    temp1=(L*AVG[servers,numSame])/servers;

    (* Test whether rho is in range *)
    If[((temp1 >= 1)||(temp1 < 0)),
        Print[" "];
        Print["<<<<<    ERROR: Resource #",i," >>>>> "];
        Print["<<<<<     Rho >=1 or Rho < 1   >>>>>>"];
        Abort[];
    ]
    ]
]
```

Step2[]:=

```
Module[{},

    For[i=1, i <= maxServers, i++,
        TWAR[i] = 0;

        (* Find cycle time for each of the resources *)
        For[j=1, j <= numServer[i], j++,
            (* Call cycle time procedure *)
            MGSCycleTime[i,j];

            (* Queue waiting time *)
            WTQ[i,j] = wqMGS;

            (* Cycle time *)
            WT[i,j] = wMGS;

            (* Compute total aggregate resource cycle time *)
            TWAR[i] = TWAR[i]+WT[i,j];

            (* Set the arrival avariability to TBA since Poisson arrival *)
            VA[i,j] = TBA^2;

            Clear[wqMGS,wMGS];
        ]

        If[(numServer[i] > 0),
           (* Compute average aggregate resource cycle time *)
           Unprotect[AAR[i]];
          temp = numServer[i];
          AAR[i] = TWAR[i]/temp;
         ]
    ]
]
```

Step2Single[]:=

```
Module[{},

    (* Compute first resource with MGS formulas *)
    MGSCycleTime[1,1];

    (* Queue waiting time *)
    WTQ[1,1] = wqMGS;

    (* Cycle time *)
    WT[1,1] = wMGS;

    (* Compute total aggregate resource cycle time *)
    TWAR[1] = WT[1,1];
    arrivalVar = TBA^2;
    VA[1,1]=arrivalVar;
    Clear[wqMGS,wMGS];

    (* Find cycle time using G/G/1 for each of the resources *)
    For[j=2, j <= numServer[1], j++,

        (* Call cycle time procedure *)
        GG1CycleTime[j,WTQ[1,(j-1)]];

        (* Queue waiting time *)
        WTQ[1,j] = wqGG1;

        (* Cycle time *)
        WT[1,j] = wGG1;

        (* Compute total aggregate resource cycle time *)
        TWAR[1] = TWAR[1]+WT[1,j];

        Clear[wqGG1,wGG1];
    ]

    (* Compute average aggregate resource cycle time *)
    Unprotect[AAR[1]];
    AAR[1]=TWAR[1]/numServer[1];
]
```

Step3[]:=

```
Module[{},

    (* Estimate the service time coefficient of variation for each AR *)
    For [i=1, i <= maxServers, i++,
       estSQCV[i] = 0;
       If[(numServer[i] > 0),
          For [j=1, j<= numServer[i], j++,
               estSQCV[i] =estSQCV[i] + ((WT[i,j] / TWAR[i])*SQCV[i,j]);
          ]
        ]
    ]

    (* Compute service mean for each aggregation resource *)
    For[i=1, i<=maxServers, i++,

          (* No need to solve if the AR consists of only 1 resource *)
        If[(numServer[i] == 1),
             MAR[i] = AVG[i,1];
        ]

        (* Solve for mean value *)
        If[(numServer[i] > 1),
             MGSServiceMean[i,estSQCV[i],AAR[i]];
             MAR[i] = meanVal;
        ]
        Clear[meanVal];
    ]
]
```

Step4[]:=

```
Module[{att},


(* Find service means for each aggregation block *)
For [i=1, i <= maxServers, i++,

 If[(numServer[i] > 0),

  (* Copy original values and save them *)
  For[j=1, j<=numServer[i],j++,
      where[i, j] = j;
      origAVG[i,j]=AVG[i,j];
      origSQCV[i,j]=SQCV[i,j];
      origWT[i,j]=WT[i,j];
  ];

  For[att=1, att<=12, att++,

    For[j=1,j<=numServer[i],j++,
        where[j]=j;
        AVG[i,j] = origAVG[i,j];
        SQCV[i,j] = origSQCV[i,j];
        WT[i,j] = origWT[i,j];
    ];

    (* att = 1 -> solve as is, distweight 1
        att = 2 -> sort alternating mean, high-low, use distweight 1
        att = 3 -> sort by highest mean, use distweight 1
        att = 4 -> sort by lowest mean, use distweight 1
        att = 5 -> sort by largest cycle time, use distrweight 1
        att = 6 -> sort by smallest cycle time, use distweight 1
        att = 7 -> solve as is, distweight 2
        att = 8 -> sort alternating mean, high-low, use distweight 2
        att = 9 -> sort by highest mean, use distweight 2
        att = 10 -> sort by lowest mean, use distweight 2
        att = 11 -> sort by largest cycle time, use distrweight 2
        att = 12 -> sort by smallest cycle time, use distweight
     *)

     If[((att != 1)&&(att != 7)), Swap[i,att]];

     If[((att == 2)||(att == 8)),  HighLow[i]];

     For[j=1, j<= numServer[i], j++,
      tAVG[i,j]=AVG[i,j];
      tSQCV[i,j]=SQCV[i,j];
      tWT[i,j]=WT[i,j];
      DWT[i,j]=0;
     ];


     (* Set global variables for recursive algorithm *)
     Unprotect[meanAR];
```

```
        meanAR = MAR[i];
        Unprotect[sumAR];
        sumAR = 1;

        (* Call the recursive algorithm - will determine AR #i's weights *)
        If[(att <= 6),
            DistWeight1[i, 1, numServer[i], numServer[i], meanAR, sumAR],
            DistWeight2[i,1,numServer[i]];
        ];

        Unprotect[fail];
        fail = 0;
        For [j=1, j<= numServer[i],j++,
            If[(DWT[i,j] < 0), fail = 1];
        ];

        (* Passes - YEA! *)
        If[(fail == 0), att = 13];
      ];
];
];
];
```

Swap[i_,type_]:=

```
Module[{Temp1},
(*Print["Enter Swap"];*)

    If[((type != 1)||(type != 7)),
     (* Perform a selection sort and swap values - original values are lost *)
      For[j=numServer[i], j >= 2, j--,
         m = 1;

       For[k=2, k<=j, k++,

        (* Sort mean largest to smallest *)
        If[((type == 2)||(type == 3)||(type == 8)||(type ==9)),
            If[(AVG[i,m] > AVG[i,k]),  m = k];
        ];

        (* Sort mean smallest to largest *)
        If[((type == 4)||(type == 10)),
            If[(AVG[i,m] < AVG[i,k]),  m = k];
        ];

        (* Sort weight largest to smallest *)
        If[((type == 5)||(type == 11)),
            If[(WT[i,m] > WT[i,k]),  m = k];
        ];

        (* Sort weight smallest to largest *)
        If[((type == 6)||(type == 12)),
            If[(WT[i,m] < WT[i,k]),  m = k];
        ];

       Unprotect[Temp1];
       Temp1=WT[i,m];
       WT[i,m] = WT[i,j];
       WT[i,j] = Temp1;

       Temp1=AVG[i,m];
       AVG[i,m ] = AVG[i,j];
       AVG[i,j] = Temp1;

       Temp1=SQCV[i,m];
       SQCV[i,m ] = SQCV[i,j];
       SQCV[i,j] = Temp1;

       Temp1=rNum[i,m];
       rNum[i,m] = rNum[i,j];
       rNum[i,j] = Temp1;

       Temp1=where[i,m];
          where[i,m] = where[i,j];
       where[i,j] = Temp1;
      ]
      ]
  ]
```

]

UserMain[]:=

```
Module[{},

      Step1[];
      If[(singleCheck == 0),
            Step2[],
            Step2Single[];
       ]

      Step3[];
      FPrint[];
      Step4[];
      APrint[];
]
```

APPENDIX C

OUTPUT OF THE AGGREGATION PROGRAM FOR THE MPD EXAMPLE

Flow Line Description.....

Resource #1 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01
Arrival Variability is 10000
Arrival COV is 1.
Arrival SQCOV is 1.
Number of Parallel Resources is 1
Service Dist is Uniform (75, 85)
Service Mean is 80.
Service Variability is 8.33333
Service SD is 2.88675
Service Time is COV is 0.0360844
Service Time SQCOV is 0.00130208
Resource Utilization is 0.8
Est. Queue Waiting Time is 160.208
Est. Resource Cycle Time is 240.208

Resource #2 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01
Arrival Variability is 10000
Arrival COV is 1.
Arrival SQCOV is 1.
Number of Parallel Resources is 2
Service Dist is Normal (130, 15)
Service Mean is 130.
Service Variability is 225.
Service SD is 15.
Service Time is COV is 0.115385
Service Time SQCOV is 0.0133136
Resource Utilization is 0.65
Est. Queue Waiting Time is 48.1872
Est. Resource Cycle Time is 178.187

Resource #3 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01
Arrival Variability is 10000
Arrival COV is 1.
Arrival SQCOV is 1.
Number of Parallel Resources is 2
Service Dist is Triag (120, 150, 180)
Service Mean is 150.

Service Variability is 150.
Service SD is 12.2474
Service Time is COV is 0.0816497
Service Time SQCOV is 0.00666667
Resource Utilization is 0.75
Est. Queue Waiting Time is 97.0714
Est. Resource Cycle Time is 247.071

Resource #4 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01
Arrival Variability is 10000
Arrival COV is 1.
Arrival SQCOV is 1.
Number of Parallel Resources is 4
Service Dist is Normal (320, 25)
Service Mean is 320.
Service Variability is 625.
Service SD is 25.
Service Time is COV is 0.078125
Service Time SQCOV is 0.00610352
Resource Utilization is 0.8
Est. Queue Waiting Time is 120.015
Est. Resource Cycle Time is 440.015

Resource #5 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01
Arrival Variability is 10000
Arrival COV is 1.
Arrival SQCOV is 1.
Number of Parallel Resources is 1
Service Dist is Triag (32, 43, 60)
Service Mean is 45.
Service Variability is 33.1667
Service SD is 5.75905
Service Time is COV is 0.127979
Service Time SQCOV is 0.0163786
Resource Utilization is 0.45
Est. Queue Waiting Time is 18.7106
Est. Resource Cycle Time is 63.7106

Resource #6 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01

Arrival Variability is 10000
Arrival COV is 1.
Arrival SQCOV is 1.
Number of Parallel Resources is 1
Service Dist is Uniform (64, 80)
Service Mean is 72.
Service Variability is 21.3333
Service SD is 4.6188
Service Time is COV is 0.06415
Service Time SQCOV is 0.00411523
Resource Utilization is 0.72
Est. Queue Waiting Time is 92.9524
Est. Resource Cycle Time is 164.952

Aggregate Flow Line Description...

Aggregate Resource #1
Arrival Mean is 100
Arrival Rate is 0.01
Total AR Cycle Time is 468.871
Average AR Cycle Time is 156.29
Est. Service Time SQCOV is 0.00434038
Est. AR Service Mean is 70.6877
Est. AR Service Rate is 0.0141467
Distribution Weights for AR #1
Weights of Resource #1 is 0.465724
Weights of Resource #5 is 0.186597
Weights of Resource #6 is 0.347679

Aggregate Resource #2
Arrival Mean is 100
Arrival Rate is 0.01
Total AR Cycle Time is 425.259
Average AR Cycle Time is 212.629
Est. Service Time SQCOV is 0.0094518
Est. AR Service Mean is 141.373
Est. AR Service Rate is 0.0070735
Distribution Weights for AR #2
Weights of Resource #2 is 0.431361
Weights of Resource #3 is 0.568639

Aggregate Resource #4
Arrival Mean is 100
Arrival Rate is 0.01
Total AR Cycle Time is 440.015

Average AR Cycle Time is 440.015
Est. Service Time SQCOV is 0.00610352
Est. AR Service Mean is 320.
Est. AR Service Rate is 0.003125
Distribution Weights for AR #4
  Weights of Resource #4 is 1

APPENDIX D

OUTPUT OF THE AGGREGATION PROGRAM FOR A SINGLE SERVER FLOW LINE

Flow Line Description.....

Resource #1 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01
Arrival Variability is 10000
Arrival COV is 1.
Arrival SQCOV is 1.
Number of Parallel Resources is 1
Service Dist is Uniform (75, 85)
Service Mean is 80.
Service Variability is 8.33333
Service SD is 2.88675
Service Time is COV is 0.0360844
Service Time SQCOV is 0.00130208
Resource Utilization is 0.8
Est. Queue Waiting Time is 160.208
Est. Resource Cycle Time is 240.208

Resource #2 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01
Arrival Variability is 3608.33
Arrival COV is 0.600694
Arrival SQCOV is 0.360833
Number of Parallel Resources is 1
Service Dist is Triag (32, 43, 60)
Service Mean is 45.
Service Variability is 33.1667
Service SD is 5.75905
Service Time is COV is 0.127979
Service Time SQCOV is 0.0163786
Resource Utilization is 0.45
Est. Queue Waiting Time is 2.87319
Est. Resource Cycle Time is 47.8732

Resource #3 of the Flow Line
Arrival Mean is 100
Arrival Rate is 0.01
Arrival Variability is 3358.62
Arrival COV is 0.579536
Arrival SQCOV is 0.335862
Number of Parallel Resources is 1
Service Dist is Uniform (64, 80)
Service Mean is 72.

Service Variability is 21.3333
Service SD is 4.6188
Service Time is COV is 0.06415
Service Time SQCOV is 0.00411523
Resource Utilization is 0.72
Est. Queue Waiting Time is 22.4827
Est. Resource Cycle Time is 94.4827

Aggregate Flow Line Description...

Aggregate Resource #1
Arrival Mean is 100
Arrival Rate is 0.01
Total AR Cycle Time is 382.564
Average AR Cycle Time is 127.521
Est. Service Time SQCOV is 0.00388349
Est. AR Service Mean is 65.4155
Est. AR Service Rate is 0.0152869
Distribution Weights for AR #1
Weights of Resource #1 is 0.0344865
Weights of Resource #2 is 0.254089
Weights of Resource #3 is 0.711425

APPENDIX E

EXPONENTIAL SERVER TEST CASES

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	7	EX(641.524)	641.524	.851779 (7)
2	6	EX(310.426)	310.426	.350778 (6)
3	6	EX(103.1111)	103.1111	.319653 (6)
4	5	EX(183.41)	183.41	.254892 (5)
5	1	EX(41.7156)	41.7156	.321298 (1)
6	4	EX(266.471)	266.471	.594674 (4)
7	5	EX(161.954)	161.954	.251310 (5)
8	5	EX(75.0584)	75.0584	.246956 (5)
9	6	EX(127.46)	127.46	.329568 (6)
10	7	EX(253.901)	253.901	.148221 (7)
11	4	EX(168.712)	168.712	.405326 (4)
12	2	EX(140.706)	140.706	1 (2)
13	8	EX(162.162)	162.162	1 (8)
14	5	EX(60.221)	60.221	.246843 (5)
15	1	EX(40.5197)	40.5197	.314838 (1)
16	1	EX(48.5339)	48.5339	.363864

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 3964.191
Average Cycle Time:	3998.93	
Variance:	825.651	
Standard Deviation:	28.7341	
Relative Error: .8763%		

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	4	E(188.996)	188.996	1 (4)
2	3	E(83.9377)	83.9377	.232116 (3)
3	3	E(99.1005)	99.1005	.328082 (3)
4	3	E(185.716)	185.716	.328082 (3)
5	8	E(460.681)	460.681	1 (8)
6	3	E(43.833)	43.833	.200618 (3)
7	5	E(32.9265)	32.9265	.0926063 (5)
8	2	E(146.099)	146.099	.37258 (2)
9	2	E(128.224)	128.224	.28291 (2)
10	5	E(464.574)	464.574	.907394 (5)
11	2	E(141.492)	141.492	.34451 (2)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 3558.436
Average Cycle Time:	3631.2	
Variance:	1142.99	
Standard Deviation:	33.8082	
Relative Error: 2.0448%		

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	3	E(58.8783)	58.8783	.447214 (3)
2	8	E(104.824)	104.824	.200541 (8)
3	8	E(679.662)	679.662	.512008 (8)
4	3	E(143.636)	143.636	.552786 (3)
5	1	E(74.4635)	74.4635	.524694 (1)
6	5	E(435.68)	435.68	.629902 (5)
7	4	E(308.982)	308.982	.143820 (4)
8	4	E(27.2446)	27.2446	.0580466 (4)
9	4	E(197.227)	197.227	.138475 (4)
10	2	E(130.989)	130.989	.151827 (2)
11	8	E(186.296)	186.296	.287451 (8)
12	6	E(181.325)	181.325	.50157 (6)
13	5	E(387.702)	387.702	.370098 (5)
14	2	E(116.68)	116.68	.127278 (2)
15	2	E(185.617)	185.617	.720896 (2)
16	4	E(349.964)	349.964	.503261 (4)
17	1	E(71.81)	71.81	.475306 (1)
18	6	E(134.895)	134.895	.49843 (6)
19	4	E(111.034)	111.034	.156398 (4)
20	7	E(92.1506)	92.1506	1 (7)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 7402.9133
Average Cycle Time:	7501.87	
Variance:	3508.6	
Standard Deviation:	59.2335	
Relative Error: 1.3367%		

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	7	E(164.999)	164.999	1 (1)
2	8	E(392.161)	192.161	.508823 (8)
3	5	E(171.873)	171.873	.502199 (5)
4	8	E(290.42)	290.42	.491177 (8)
5	4	E(220.934)	220.934	1 (4)
6	5	E(156.27)	156.27	.497801 (5)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 1433.636
Average Cycle Time:	1433.77	
Variance:	4.11609	
Standard Deviation:	2.02882	
Relative Error: .0093%		

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	8	E(613.651)	613.651	.612957 (8)
2	6	E(284.327)	284.327	.177185 (6)
3	5	E(428.332)	428.332	1 (5)
4	3	E(168.152)	168.152	.257442 (3)
5	1	E(68.1015)	68.1015	.487263 (1)
6	1	E(42.53)	42.53	.270454 (1)
7	1	E(35.8477)	35.8477	.242283 (1)
8	8	E(403.528)	403.528	.387043 (8)
9	3	E(85.1781)	85.1781	.188898 (3)
10	4	E(304.81)	304.81	.511806 (4)
11	7	E(536.997)	536.997	1 (7)
12	4	E(175.376)	175.376	.256852 (4)
13	6	E(529.389)	529.389	.645171 (6)
14	6	E(191.892)	191.892	.177644 (6)
15	3	E(200.374)	200.374	.332109 (3)
16	3	E(136.342)	136.342	.221552 (3)
17	4	E(63.6065)	63.6065	.231342 (4)
18	2	E(137.432)	137.432	1 (2)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 6255.320
Average Cycle Time:	6312.8	
Variance:	696.924	
Standard Deviation:	26.3993	
Relative Error: .9189%		

Type of Test: **Exponential Servers** Test Case Number: **6** 7

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	E(40.4746)	40.4746	.601017 (1)
2	6	E(302.213)	302.213	.520419 (6)
3	1	E(10.3324)	10.3324	.398983 (1)
4	8	E(539.917)	539.917	.306553 (8)
5	2	E(21.0607)	21.0607	1 (2)
6	8	E(645.478)	645.478	.463343 (8)
7	7	E(76.2595)	76.2595	1 (7)
8	6	E(182.238)	182.238	.479581 (6)
9	8	E(126.784)	126.784	.230103 (8)
10	5	E(311.865)	311.865	1 (5)
11	4	E(265.834)	265.834	1 (4)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 2926.512
Average Cycle Time:	2946.87	
Variance:	78.7402	
Standard Deviation:	8.87357	
Relative Error: .6956%		

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	7	E(470.654)	470.654	.558678 (7)
2	6	E(220.377)	220.377	.336040 (6)
3	4	E(105.943)	105.943	.09910831 (4)
4	3	E(109.351)	109.351	.347358 (3)
5	8	E(649.843)	649.843	.418376 (8)
6	3	E(70.1249)	70.1249	.322024 (3)
7	5	E(348.024)	348.024	.593461 (5)
8	8	E(694.79)	694.79	.581624 (8)
9	1	E(46.1787)	46.1787	.181615 (1)
10	6	E(162.55)	162.55	.330655 (6)
11	2	E(130.033)	130.033	.371503 (2)
12	5	E(232.932)	232.932	.406539 (5)
13	2	E(135.876)	135.876	.403708 (2)
14	6	E(191.226)	191.226	.333305 (6)
15	4	E(359.396)	359.396	.595162 (4)
16	4	E(291.785)	291.785	.330617 (3)
17	3	E(83.759)	83.759	.330617 (3)
18	2	E(78.1607)	78.1607	.22479 (2)
19	1	E(88.0561)	88.0561	.818355 (1)
20	7	E(335.127)	335.127	.441322 (7)
21	4	E(23.584)	23.854	.10113 (4)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 7349.53
Average Cycle Time:	7468.73	
Variance:	2320.96	
Standard Deviation:	48.1764	
Relative Error: 1.6219%		

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	7	E(495.698)	495.698	.182187 (7)
2	3	E(117.996)	117.996	.177563 (3)
3	7	E(140.256)	140.256	.128570 (7)
4	3	E(230.082)	230.082	.406138 (3)
5	7	E(334.679)	334.679	.139292 (7)
6	2	E(123.768)	123.768	1 (2)
7	8	E(312.374)	312.374	.499315 (8)
8	6	E(101.246)	101.246	.261468 (6)
9	4	E(367.38)	367.38	.893172 (4)
10	7	E(549.771)	549.771	.252735 (7)
11	3	E(186.345)	186.345	.255546 (3)
12	3	E(72.7947)	72.7947	.161753 (3)
13	7	E(161.871)	161.871	.147352 (7)
14	6	E(410.749)	410.749	.371133 (6)
15	8	E(323.68)	323.68	.500685 (8)
16	6	E(367.965)	367.965	.367399 (6)
17	5	E(178.907)	178.907	1 (5)
18	4	E(69.1653)	69.1653	.106828 (4)
19	7	E(275.838)	275.838	.149863 (7)
20	1	E(36.9879)	36.9879	1 (1)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 7402.9133
Average Cycle Time:	7501.87	
Variance:	3508.6	
Standard Deviation:	59.2335	
Relative Error: 1.3367%		

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	4	E(185.463)	185.463	1 (4)
2	5	E(183.962)	183.962	1 (5)
3	6	E(427.187)	427.187	1 (6)
4	8	E(352.682)	352.682	1 (8)
5	1	E(71.9968)	71.9968	.719422 (1)
6	3	E(101.039)	101.039	1 (6)
7	7	E(472.028)	472.028	1 (7)
8	1	E(28.1978)	28.1978	.280578 (1)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	Average Cycle Time: 2182.571
Average Cycle Time:	2202.83	
Variance:	33.523	
Standard Deviation:	5.7899	
Relative Error: .9282%		

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	2	E(76.0439)	76.0439	1 (2)
2	1	E(25.7639)	25.7639	.325856 (1)
3	7	E(523.191)	523.191	.326682 (7)
4	7	E(463.291)	463.291	.262130 (7)
5	4	E(135.344)	135.344	.119145 (4)
6	1	E(9.20471)	9.20471	.266427 (1)
7	7	E(120.806)	120.806	.205178 (7)
8	3	E(107.919)	107.919	.0998117 (3)
9	3	E(100.199)	100.199	.097916 (3)
10	8	E(477.738)	477.738	.455455 (8)
11	3	E(277.263)	277.263	.721439 (3)
12	7	E(212.576)	212.576	.206011 (7)
13	4	E(341.757)	341.757	.443696 (4)
14	5	E(338.501)	338.501	.163881 (5)
15	1	E(40.669)	40.669	.407717 (1)
16	8	E(523.321)	523.321	.544545 (8)
17	3	E(77.6403)	77.6403	.0808333 (3)
18	4	E(335.98)	335.98	.437159 (4)
19	5	E(469.706)	469.706	.836119 (5)

Aggregate Simulation Results **Steady-State Result**

Number of Runs:	30	
Average Cycle Time:	7468.73	Average Cycle Time: 7349.53
Variance:	2320.96	
Standard Deviation:	48.1764	
Relative Error: 1.6219%		

APPENDIX F

SINGLE SERVER TEST CASES

Type of Test: **Single Servers** Test Case Number: **1**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	TR(6.03438,7.37826,38.9414)	17.418	.143870 (1)
2	1	LN(36.9703,3.5081)	36.9703	.166009 (1)
3	1	UN(29.8977,39.1813)	34.5395	.0392006 (1)
4	1	EX(31.008)	31.008	.292399 (1)
5	1	UN(33.7436,39.5302)	36.6369	.219948 (1)
6	1	LN(56.2609,1.60176)	56.2609	.138574 (1)

Aggregate Simulation Results **Full Simulation Results**

Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	284.34	Average Cycle Time:	266.06
Variance:	.60731	Variance:	.190069
Standard Deviation:	.779301	Standard Deviation:	.435969
Relative Error: 6.8706%			

Type of Test: **Single Servers** Test Case Number: **2**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	UN(52.6808,69.1178)	60.8993	.125833 (1)
2	1	EX(21.5976)	21.5676	.134422 (1)
3	1	RN(28.9176,1.65069)	28.9176	.119716 (1)
4	1	RN(35.8158,.58196)	35.8158	.416657 (1)
5	1	TR(3.85609,10.5231,14.2538)	9.5443	.0681083 (1)
6	1	UN(43.679,47.7504)	45.7147	.0531856 (1)
7	1	UN(45.9126,60.7263)	53.3194	.0820775 (1)

Aggregate Simulation Results **Full Simulation Results**

Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	341.947	Average Cycle Time:	313.707
Variance:	.397057	Variance:	.275506
Standard Deviation:	.630125	Standard Deviation:	.524886
Relative Error: 9.0020%			

Type of Test: **Single Servers** Test Case Number: **3**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	EX(75.167)	75.167	.201597 (1)
2	1	RN(62.0906,1.2018)	62.0906	.146909 (1)
3	1	LN(76.2963,4.58497)	76.2963	.133501 (1)
4	1	RN(24.1441,.64017)	24.1441	.0982238 (1)
5	1	EX(70.0376)	70.0376	.109578 (1)
6	1	LN(36.0415,2.89278)	36.0415	.123802 (1)
7	1	RN(53.8442,4.40897)	53.8442	.0338425 (1)
8	1	EX(38.5093)	38.5093	.152546 (1)

Aggregate Simulation Results **Full Simulation Results**

Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	957.28	Average Cycle Time:	938.357
Variance:	77.4338	Variance:	46.7232
Standard Deviation:	8.7997	Standard Deviation:	6.83544
Relative Error: 2.0166%			

Type of Test: **Single Servers** Test Case Number: **4**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	TR(13.2021,14.3314,29.062)	18.8652	.0844431 (1)
2	1	EX(67.6241)	67.6241	.214037 (1)
3	1	TR(6.49979,7.09306,91.9114)	35.1681	.0985627 (1)
4	1	LN(22.7481,1.02535)	22.7481	.0881934 (1)
5	1	EX(53.7082)	53.7082	.0583586 (1)
6	1	EX(49.96)	49.96	.148159 (1)
7	1	TR(17.0748,21.1078,39.0845)	25.7557	.125346 (1)
8	1	TR(33.5248,68.6314,87.2534)	63.1365	.1829 (1)

Aggregate Simulation Results		**Full Simulation Results**	
Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	684.47	Average Cycle Time:	654.05
Variance:	18.6194	Variance:	10.2647
Standard Deviation:	4.31502	Standard Deviation:	3.20385
Relative Error: 4.6510%			

Type of Test: **Single Servers** Test Case Number: **5**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	RN(25.1093,2.33693)	25.1093	.071339 (1)
2	1	RN(24.2974,.991618)	24.2974	.116718 (1)
3	1	LN(49.3307,1.34513)	49.3307	.188758 (1)
4	1	LN(63.8064,3.33453)	63.8084	.081345 (1)
5	1	EX(62.045)	62.045	.136191 (1)
6	1	EX(41.4581)	41.4581	.236257 (1)
7	1	UN(58.1551,72.0328)	65.094	.0492735 (1)
8	1	UN(42.0843,52.6536)	47.3689	.120123 (1)

Aggregate Simulation Results **Full Simulation Results**

Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	612.1	Average Cycle Time:	590.79
Variance:	7.84828	Variance:	3.28162
Standard Deviation:	2.80148	Standard Deviation:	1.81152
Relative Error: 3.6070%			

Type of Test: **Single Servers** Test Case Number: **6**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	UN(37.2913,40.3897)	38.8405	.110414 (1)
2	1	EX(50.1321)	50.1321	.107848 (1)
3	1	LN(53.8684,4.58105)	53.8684	.160653 (1)
4	1	UN(69.2811,74.3105)	71.7958	.0793943 (1)
5	1	TR(20.3,28.4779,50.2513)	33.0097	.132721 (1)
6	1	RN(64.4865,3.53079)	64.4865	.0428772 (1)
7	1	RN(72.1901,3.86079)	72.1901	.063162 (1)
8	1	TR(9.3663,34.0359,67.2058)	36.8693	.265565 (1)
9	1	EX(57.1739)	57.1739	.0373651 (1)

Aggregate Simulation Results		**Full Simulation Results**	
Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	677.427	Average Cycle Time:	676.797
Variance:	2.19099	Variance:	2.22585
Standard Deviation:	1.4802	Standard Deviation:	1.49193
Relative Error: .0931%			

Type of Test: **Single Servers** Test Case Number: **7**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	TR(6.73922,10.0269,15.1726)	10.6463	.085642 (1)
2	1	EX(38.882)	38.882	.129118 (1)
3	1	TR(7.02216,16.2104,26.0004)	16.411	.0926933 (1)
4	1	EX(68.167)	68.167	.24496 (1)
5	1	LN(39.8866,1.6883)	39.8866	.0676431 (1)
6	1	UN(34.4051,35.7375)	35.0713	.109277 (1)
7	1	RN(77.5448,5.28714)	77.5448	.110844 (1)
8	1	TR(21.5809,49.4403,84.5969)	51.8727	.159823 (1)

Aggregate Simulation Results **Full Simulation Results**

Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	691.49	Average Cycle Time:	643.93
Variance:	11.0416	Variance:	9.87597
Standard Deviation:	3.32289	Standard Deviation:	3.1426
Relative Error: 7.3859%			

Type of Test: **Single Servers** Test Case Number: **8**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	TR(11.4769,16.802,41.894)	23.391	.211893 (1)
2	1	TR(6.7004,27.2152,31.2644)	21.7267	.162140 (1)
3	1	RN(29.7351,1.71058)	29.7351	.149073 (1)
4	1	TR(7.36975,9.48395,32.2309)	16.3615	.123412 (1)
5	1	EX(35.1738)	35.1738	.142700 (1)
6	1	UN(35.3489,37.5103)	36.4296	.111561 (1)
7	1	TR(5.93027,7.74329,10.6431)	8.10556	.0992203 (1)

Aggregate Simulation Results **Full Simulation Results**

Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	214.98	Average Cycle Time:	203.47
Variance:	.261655	Variance:	.0545862
Standard Deviation:	.511522	Standard Deviation:	.233637
Relative Error: 5.6569%			

Type of Test: **Single Servers** Test Case Number: **9**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	TR(11.0186,17.0233,66.269)	31.437	.0812232 (1)
2	1	UN(42.3664,47.2833)	44.8249	.0810911 (1)
3	1	TR(14.197,30.0482,40.8699)	28.3717	.275825 (1)
4	1	RN(59.6102,4.03677)	59.6102	.253205 (1)
5	1	LN(29.4613,1.06546)	29.4613	.308655 (1)

Aggregate Simulation Results **Full Simulation Results**

Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	258.057	Average Cycle Time:	239.94
Variance:	.2779782	Variance:	.152138
Standard Deviation:	.528944	Standard Deviation:	.390049
Relative Error: 7.5506%			

Type of Test: **Single Servers** Test Case Number: **10**

Resource #	# Servers	Distribution	Mean	Weight (AR)
1	1	TR(37.4795,48.9889,74.2355)	53.658	.0762831 (1)
2	1	TR(14.6084,15.4301,44.7675)	24.9353	.0569433 (1)
3	1	RN(60.6976,1.1293)	60.6976	.0666616 (1)
4	1	LN(23.324,1.18517)	23.324	.0662817 (1)
5	1	RN(28.5769,1.29901)	28.5769	.0802676 (1)
6	1	EXPON(78.5979)	78.5979	.150786 (1)
7	1	TR(2.38702,5.48268,16.7479)	8.20586	.052658 (1)
8	1	EX(73.075)	73.075	.157072 (1)
9	1	TR(18.7301,25.2577,74.6774)	39.555	.159243 (1)
10	1	UN(49.1881,67.4646)	58.3264	.133804 (1)

Aggregate Simulation Results **Full Simulation Results**

Number of Runs:	30	Number of Runs:	30
Average Cycle Time:	1012.4	Average Cycle Time:	993.51
Variance:	36.5241	Variance:	54.6989
Standard Deviation:	6.04352	Standard Deviation:	7.39587
Relative Error: 1.90134%			