A FRAMEWORK FOR THE ANALYSIS OF MANUFACTURING SYSTEMS CONTROLLED BY A PRODUCTION AUTHORIZATION CARD SCHEME

by

Corinne MacDonald

Submitted
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Major Subject: Industrial Engineering

at

DALHOUSIE UNIVERSITY

Halifax, Nova Scotia February, 2006

Your file Votre référence
ISBN: 978-0-494-20558-7
Our file Notre référence
ISBN: 978-0-494-20558-7

DALHOUSIE UNIVERSITY

To comply with the Canadian Privacy Act the National Library of Canada has requested that the following pages be removed from this copy of the thesis:

Preliminary Pages
- Examiners Signature Page
- Dalhousie Library Copyright Agreement

Appendices
- Copyright Releases (if applicable)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ANN	Artificial Neural Network
BP	Backpropagation
CONWIP	Constant Work in Process
CI	Confidence Interval
CT	Cycle Time
EA	Evolutionary Algorithm
EKCS	Extended Kanban Control System
FGI	Finished Goods Inventory
FIFO	First-In, First-Out priority scheme
GKCS	Generalized Kanban Control System
IPA	Infinitesimal Perturbation Analysis
LCL	Lower Confidence Limit (lower bound of confidence interval)
MAE	Mean Absolute Error
ME	Mean Error
MRP	Material Requirements Planning
MSE	Mean Squared Error
MTTR	Mean Time to Repair
NN	Neural Network
OPT	Optimized Production Technology
PA	Production Authorization
PAC	Production Authorization Card
RSM	Response Surface Methodology
SA	Simulated Annealing
SPT	Shortest Processing Time
UCL	Upper Confidence Limit (upper bound of confidence interval)
WIP	Work in Process Inventory
k_i	Number of Process Tags at Cell i
r_i	Batching parameter at Store i
z_i	Initial Inventory Level at Store i
τ_i	Delay time Between Orders and Requisitions from Cell i

ACKNOWLEDGEMENTS

This opportunity to pursue a Ph.D. came about as the result of a fairly unique opportunity to teach as a full-time lecturer in the department. My work on this thesis was often carried out on a part-time basis, due to my other responsibilities, and the road was not an easy one. However, there were many people who helped and encouraged me along the way, and I owe all of them a debt of gratitude.

First, my supervisor, Dr. Eldon Gunn, who originally offered me this opportunity, was a constant source of encouragement, inspiration, and motivation. Through financial support from Dr. Gunn's NSERC research grant, I was able to present my work at conferences such as the MOPTA 2004 conference and the 2005 Winter Simulation Conference, as well as obtain the necessary computer hardware and software.

I would like to acknowledge Dr. John Buzacott and Dr. George Shanthikumar, who introduced the Production Authorization Card (PAC) system, and Dr. Krystyna Bielunska-Perlikowski, who first developed a simulation model of a manufacturing system operating with the PAC system, as the inspiration for this work. It was Dr. Bielunska-Perlikowski's original computer code that formed the basis of the simulation model developed in this thesis.

All the faculty members in the Department of Industrial Engineering made me feel welcome and valued from my first day, and provided me with assistance whenever I needed it. Many thanks are also extended to my supervisory committee for their feedback and encouragement.

Last, but certainly not least, I would like to thank my family and friends for their constant support and encouragement, and in particular, my husband Glen and my children for their love and support.

ABSTRACT

The selection of a strategy to control the flow of information and materials in a manufacturing system is an important problem. Buzacott and Shanthikumar (1992) presented a modeling framework, the Production Authorization Card Scheme (PAC), to encompass several different traditional control strategies, such as Kanban, CONWIP, and Base Stock Systems. The challenge they presented was to develop an analysis framework to enable the determination of the best control strategy for a manufacturing system, in terms of the performance measures of such a system, such as average inventory and customer service levels.

While simple systems operating under the PAC scheme may be analyzed analytically under very simplifying assumptions, complex systems do not permit this analysis. Thus, simulation is necessary to study the effects of PAC parameter settings on system performance. In this thesis, we discuss the PACSIM simulation model, which we designed to estimate several performance measures for systems with complexities such as multi-product systems, setup times, cells with multiple servers, and assembly cells.

With the ability to estimate this performance, the problem now becomes the ability to determine the PAC system parameters which provide the best operating strategy in terms of the system performance measures. This would suggest the use of simulation optimization techniques. However, we believe that simulation optimization is inappropriate for this framework. We argue that the correct approach is simulation metamodelling, which is an attempt to approximate the expected value functions of the system performance, with respect to the PAC scheme parameters. Such metamodels provide the ability to apply deterministic optimization techniques, as well as the ability to explore trade-offs amongst the various performance measures, so that the best control strategy may be selected based on policies and outside decision factors not easily integrated into any optimization approach.

Of the metamodel approaches available, neural networks present many attractive advantages. In this thesis, we demonstrate that feedforward neural networks are a highly practical and feasible approach, even for large numbers of input parameters. One key to training reasonably accurate neural networks is efficient simulation experimental design. We present a sampling strategy to with good space-filling properties, and a set of rules designed to ensure that the PAC parameters included in the input space are feasible and reasonable for future analysis purposes.

We demonstrate the potential of this analysis framework on several manufacturing examples. The systems we selected have from four to ten input parameters. We illustrate with these systems some of the types of analysis permitted by this framework.

CHAPTER 1

INTRODUCTION

The design and analysis of production systems is one of the oldest problems in industrial engineering (Altiok and Stidham, 1983). The choice of a manufacturing control strategy for a given manufacturing system design is one component of this problem. Control strategies or coordination schemes, such as MRP, CONWIP, Kanban and Base Stock systems, are put in place to control the flow of material and coordinate the flow of information in the manufacturing system, in order to control the performance of the system.

In 1992, Buzacott and Shanthikumar commented on previous studies on the various manufacturing coordination schemes found in the literature:

> "None of these studies, however, has resulted in (A) providing a basic framework under which all of these approaches can be systematically compared nor in (B) choosing the coordination scheme most appropriate for the manufacturing system at hand". (Buzacott and Shanthikumar, 1992, p. 36)

They introduced a Production Authorization Card (PAC) coordination scheme for material and information flow control in multiple-cell manufacturing systems. The PAC scheme involves setting the right levels of inventory, the number of PACs for each cell, along with the rules for card transmittal between cells (Buzacott and Shanthikumar, 1993). They further demonstrated that traditional mechanisms such as MRP, Kanban, OPT, and CONWIP could be seen as special instances of their scheme. This new scheme "provides a framework for developing coordination and control mechanisms that combine the desirable features of more than one of these traditional approaches." (Buzacott and Shanthikumar, 1992, p. 36).

Buzacott and Shanthikumar (1992, 1993) presented the means to analytically determine the optimal PAC parameters for a single-cell system and for multiple cell series systems under special assumptions, such as exponentially distributed processing

times. However, they left the development of general performance models and subsequent selection of the optimal PAC parameters to future research. They wrote

> "...we do not provide an approach to the selection of the parameters for the optimized system.
>
> To obtain the best possible coordination system so as to maximize the benefits (taking into account the inventory carrying cost), one needs to obtain the optimal parameter values...This requires two steps: development of a performance evaluation model and development of an optimization model...no general model has been developed so far. It is important that general performance models incorporating the PAC scheme be developed. It is of equal importance that we develop optimization models and efficient solution procedures to obtain the optimal parameter values for the PAC scheme". (Buzacott and Shanthikumar,1992, pp. 49-50)

This thesis presents a unified framework for the modeling, analysis, and subsequent determination of appropriate control strategy for complex manufacturing systems. The Production Authorization Card (PAC) Scheme proposed by Buzacott and Shanthikumar (1992), a general performance simulation model of systems operating under the PAC scheme (PACSIM), and neural network metamodels form the basis of this framework.

1.1 Manufacturing Control Strategies

For a discrete manufacturing system, raw materials are converted into finished products through a series of processes carried out at one or more work centres or cells in the system. Cells in the system require raw materials and/or *component parts* produced by another cell in the system. The time involved to complete processing at a cell is usually variable. The flow of the parts through the system starts at some initial cell (upstream) and flows downstream to subsequent cells along a predetermined route until finally arriving at a finished goods inventory point. Along this route, component parts may be immediately sent to the next cell for further processing, or temporarily stored until requested by a downstream cell. Individual cells may be free to produce if the materials required for the process are available, or they may be required to wait until they receive authorization to do so. Upon receipt of this authorization, the producing cell will request the required materials and components to be delivered from supplying stores (or

simply remove them). If requested raw materials and/or component parts arrive at a cell for further processing and there is already work in process at the station, they join a queue and wait for an available resource. Demand for the finished products may come in the form of actual orders from customers, or by orders generated by management to meet a forecast of future demand. How and when information about this demand is communicated through the system, and when and why individual cells receive authorization to produce a part or product is determined by a control strategy.

The purpose of a control strategy is to limit the amount of work in process (WIP) while ensuring adequate throughput and customer service levels. Little's Law shows a direct relationship between the amount of WIP in the system and the product of throughput and cycle time (see Hopp and Spearman, 2001). If WIP is kept too low, throughput will be low, as cells wait for work to do. If WIP is too high, throughput will increase to its maximum theoretical value, but at the expense of an increase in cycle time and holding costs. Various control strategies have been developed to allow management to control the production in order to meet planned orders or finished goods replenishment requirements.

Materials Requirements Planning (MRP) is a strategy where management uses a forecast to predict the future demand. For each finished product, an estimate of the lead time (time for a product to be produced by the system once it is started) is required. Jobs are 'released' to the system using this lead time estimate; the goal is then to have the finished product available in time to meet the anticipated demand. MRP is considered a push strategy, so called because once work is released to the system, it is pushed through the system; stations produce product provided there is work available to do, and the amount of work released to the system depends on the forecast, not on the number of jobs already in the system.

Pull strategies authorize production in the system based on the current state of the system. These strategies respond to demand by authorizing new work only when an actual demand materializes. Kanban strategies limit the amount of work in process at a station by using a set number of Kanban cards for each product produced at the station.

When a product is requested from a downstream cell, a Kanban card is pulled and sent to the station, thus authorizing the station to produce a replacement. If an order arrives to find no Kanban cards available, no authorization is created until a Kanban card is returned to the store, which will occur upon completion of a product at the cell. Therefore, the number of authorizations in process at the cell (jobs in the queue) cannot exceed the set number of Kanban cards for the product. During this time, no information about the waiting order is communicated further upstream. The number of Kanban cards must be determined for each part or product produced at each station.

CONWIP (Hopp and Spearman, 2001) strategies involve having a constant amount of work in process for the entire system; when a customer order is filled with a product from finished goods, an authorization is sent to the first cell(s) of the process to start production of another unit. All cells downstream from the start of the process are authorized to produce product whenever the required components are available.

A Base Stock system involves setting an inventory goal at each store supplied by a production station; whenever product is removed from the store for processing further downstream, an authorization to produce a replacement is provided to the cell. This strategy differs from a Kanban system in that if the inventory at the store is depleted and further requests arrive from downstream, authorizations are also generated to fill these requests. Any time the amount of inventory in the store reaches the base stock level, it means that there are no authorizations in process at the station.

There has been a great deal of literature evaluating and comparing these and other strategies, leading to some generally accepted rules regarding the choice of strategy; for example, Kanban systems are expected to work well only in situations where demand is relatively constant (Askin and Goldberg, 2002). Even when only one type of strategy is being considered, the determination of the best parameters for that strategy is not straightforward; for example, for a Kanban system, one must decide on the number of Kanban cards to use at each station. Recently, there has been some attention paid to hybrid strategies, which are usually derivations or combinations of the traditional

strategies. Examples are a hybrid MRP/Kanban system (Deleersnyder et al., 1992) and the Extended Kanban Control System (Dallery and Liberopoulos, 2000).

1.2 Performance Models of Manufacturing Systems

One of the difficulties in evaluating manufacturing control strategies is that exact methods for calculating the performance of these complex manufacturing systems do not exist, except for simple cases such as the two stage tandem production line, under quite restrictive assumptions.

For a two stage tandem production line operating under a push control system, if the processing times at each cell are assumed to be exponentially distributed, the arrivals to the line are a Poisson process or are accepted freely into the system by the first station, there is only one machine (or server) at each cell, and the buffer levels are infinite, then the output of each station (which is the input to the next station) will also be a Poisson process. Therefore, the system can be easily modeled as a series of M/M/1 queues, and the average performance measures can be calculated from basic queuing theory (e.g. Kleinrock, 1975). The interdeparture times of finished products from such a system, regardless of the number of stations, will also be exponentially distributed (Ross, 1997).

For lines where the buffer levels are finite (pull systems), researchers have used Markov chain models in an attempt to determine the average performance measures. Unfortunately, the system state space becomes so large as the number of stations or buffer levels increase that the model becomes mathematically intractable. While many models of queuing networks are decomposable (portions of the system can be treated as though they are isolated from other portions, with simple relationships among the small models), Markov chain models of transfer lines do not have this property (Gershwin, 1994). Therefore, approximate decomposition methods have been developed (e.g. Buzacott, 1972; Gershwin, 1987, Dallery, David and Xie, 1989). While these decomposition models have been shown to be good approximations, they are only applicable under certain limiting assumptions, and only for relatively simple systems. Due to all of these issues, simulation is often used to estimate the system performance.

We are interested in the ability to analyze more complex systems, with complexities such as assembly or disassembly cells, setups, transportation times, and individual stations capable of producing multiple parts. Given the difficulties in analytically modeling much simpler systems, a simulation model will be used.

1.3 Investigating Control Strategies for a Manufacturing System

In the literature, there appear to be three approaches to investigating, analyzing and choosing an optimal control strategy for a manufacturing system. The first approach is to first limit the type of control to a traditional type strategy, and then find the optimal parameters for that particular strategy (see Figure 1.1).

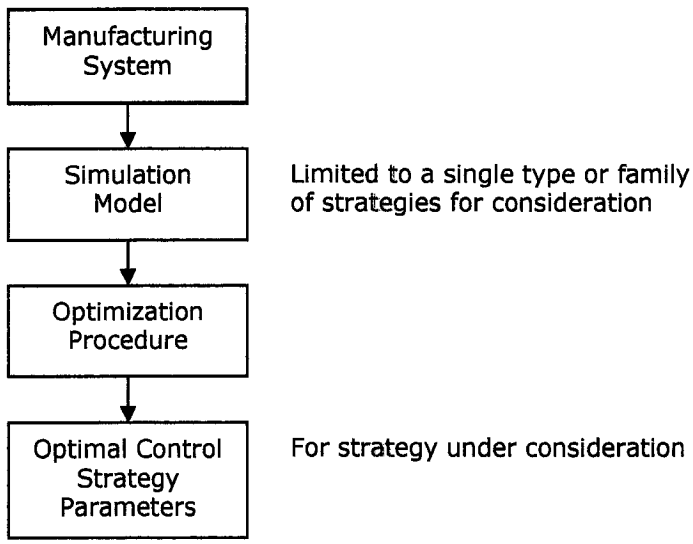

Figure 1.1 Flowchart for a Common Approach to Control Strategy Analysis

For example, Akturk and Erhun (1999) provide an overview of literature on designing optimal Kanban systems. Most researchers use simulation models of a manufacturing system operating under this strategy, and the focus is to develop a methodology to find the optimal parameter settings for the strategy chosen. In the case of Kanban systems, the goal is to determine the optimal number of Kanban cards at each station, so as to achieve some goal such as minimizing cost for inventory and customer service.

Another subset of the literature is dedicated to contrasting different traditional strategies – such as push versus pull (for example, Grosfeld-Nir et al., 2000) – to determine which is the best approach for a given system; the approach, however, is similar to the first approach because the optimal settings for each strategy must be determined in order to compare the best of both. Because of the type of modeling used in each case, the analysis does not explore possible hybrid solutions – combinations or variants of traditional strategies – which may provide even better solutions. Bonvik et al. (1997) do include a defined hybrid policy, along with Kanban, CONWIP, base stock, and minimal blocking, and then, similar to the approach above, find the parameters which optimize the performance of each individual strategy and then compare the results.

Another approach, less often found in the literature, is to determine the appropriate type of strategy using the characteristics of the system itself. For example, Huang and Kusiak (1998) developed a methodology for developing a control system which pushes through certain manufacturing stages and pulls elsewhere based on the characteristics and value added at these stages. The methodology involves labelling each stage; for example, a critical stage lies along the critical path, and a bottleneck stage has the highest utilization. Their next step is to assign either a push or pull strategy to each stage based on rules they developed, such as push at the critical stages, or pull up to a bottleneck stage. For stages with multiple labels, the implementation priority of the rules depends on the business objective, since each rule accomplishes a particular objective (reduces WIP, shortens delivery lead time, etc.). They then develop different simulation scenarios, with different buffer sizes or processing time distributions, and compare these with a simulation of the system operating under a total push system, to show that their methodology results in improved performance. Gaury et al. (2001) term Huang and Kusiak's system a *customized* control system.

1.4 Research on PAC Schemes and Other Unified Frameworks

There has been some limited work on developing unified frameworks for the study of manufacturing control strategies. The Generalized Kanban Control System (GKCS)

(Frein et al., 1995) may be used to represent pull-type strategies, where Kanban, CONWIP and Base Stock may be seen as special instances of the scheme. It is similar to the PAC scheme; however, the PAC scheme also allows for batch processing, and for time-delay release of orders based on forecasts of demand, if available. The Extended Kanban Control System (EKCS) (Dallery and Liberopoulos, 2000) is a combination of Kanban and Base Stock systems, and also includes both of these as special cases. The EKCS may be extended to include batch production and time delays in the issuance of demand notices to queues.

Gaury et al. (2001) developed a pull control system for a given manufacturing line, without *a priori* limiting of the control to traditional pull systems. Their design included Kanban, CONWIP and Hybrid; local and integral control; and segmented and joint systems. Their methodology simultaneously determines the type of policy, which may be a hybrid policy, and the parameters of that policy which produce optimal performance. Simulation and an evolutionary algorithm are used to find the policy which minimizes WIP while maintaining a minimum customer service percentage. The methodology was tested on tandem production lines with four to eight stations. They acknowledge that more research is needed to extend their methodology to other types of production systems, such as multi-product lines and assembly or disassembly systems.

Liberopoulos and Dallery (2000) have developed a unified framework for defining pull-type control mechanisms. Four basic stage coordination systems (base stock, Kanban, GKCS, and EKCS) are presented under a unified definition, and then on top of these they superimpose a local mechanism to control WIP within each stage. Some other production strategies, such as a CONWIP/Kanban hybrid control system, were shown to be special cases of their definition. However, they conclude that "considerable work needs to be done to be able to evaluate the performance of and optimize such systems" (Liberopoulous and Dallery, 2000, p. 350). Very recently, Bollon et al. (2004) presented a formulation using this framework which may be used for performance evaluation or optimization, but only for serial, single product systems.

To our knowledge, the only attempt to extend the work of Buzacott and Shanthikumar was by Bielunska-Perlikowski (1997) (Bielunska-Perlikowski and Gunn, 2002), who developed a general performance model for a manufacturing system operating under the PAC scheme using simulation. This model is able to simulate systems with assembly stations, multi-product stations, setup times and travel times. To address the problem of finding the optimal PAC parameters for a simulated manufacturing system, she developed a rudimentary simulation optimization approach. This approach required that costs be assigned to the system performance measures, such as work-in-process (WIP) inventory and customer service, in order to create a single objective function to optimize. As we will see, this approach has several problems, including the requirement to identify such costs, as well as the issues with the simulation optimization procedure itself.

1.5 Modeling and Analysis Framework for Manufacturing Systems using Neural Networks and the PAC Scheme

Buzacott and Shanthikumar (1992) introduced the Production Authorization Card (PAC) Scheme as a means to model multi-station manufacturing systems and the coordination scheme (or control strategy) used to control how information is communicated and how materials and parts flow within the system. Several traditional control strategies, such as Kanban and CONWIP, can be viewed as special instances of the PAC scheme. This makes the PAC scheme an attractive modeling approach, as it is possible to simultaneously study more than one traditional strategy or even a combination of these strategies using a single model. This coordination scheme may be employed in complex manufacturing systems, with assembly stations and multiple products. It is therefore the basis of our analysis framework. A description of the PAC scheme is provided in Chapter 2. Bielunska-Perlikowski's (1997) (Bielunska-Perlikowski and Gunn, 2002) original PAC simulation model has been updated for our purposes, and a description of this new model, PACSIM, is also included in Chapter 2.

Buzacott and Shanthikumar's (1992) objective A, mentioned at the beginning of this chapter, was to provide a framework for comparing different strategies simultaneously; this is achieved by the use of the PAC scheme and the PACSIM performance model. We then explored various options in order to achieve objective B: to determine the most appropriate control strategy for a manufacturing system. In order to achieve this, we felt it important to understand the trade-offs amongst the performance measures for a manufacturing system, such as the relationship between average inventory levels and customer service measures. With a simulation as our performance model, we investigated various simulation optimization and simulation metamodelling techniques found in the literature. These are discussed in detail in Chapter 3, as well as the reasons why we determined that, for this problem, simulation metamodelling was the best approach. The advantages of such metamodels include easier, faster evaluation than running the simulation model itself, and the ability to apply deterministic optimization techniques to find, for a given system, the appropriate control parameters. Our choice of metamodelling approach, Neural Networks, is described in Chapter 4, along with the issues in developing such metamodels from simulation data.

We discuss in detail the application of this framework in Chapter 5. Included are examples of how the metamodels developed for a system may be used to gain a better understanding of the system, through trade-off curves, deterministic optimization with constraints, and other approaches. In Chapter 6 we provide examples of the use of this framework through the analysis of several example systems, and show that the neural networks provide a flexible means of performance analysis. Finally, Chapter 7 concludes with a summary of this work and opportunities for future research.

CHAPTER 2

PRODUCTION AUTHORIZATION CARD (PAC) COORDINATION SCHEME

This chapter provides an explanation of the PAC scheme, which is the basis for our framework. Section 2.1 provides an overview of the PAC scheme, and Section 2.2 describes how traditional control strategies such as Kanban and CONWIP are special instances of this scheme, thereby enabling the simultaneous evaluation of several traditional strategies with the use of a single PAC model. Some issues with the use of the PAC scheme are discussed in Section 2.3. The PACSIM model, a simulation model designed to simulate the operation of a multi-cellular manufacturing system operating under the PAC scheme, is described in Section 2.4.

2.1 An Overview of the PAC Scheme

The Production Authorization Card (PAC) scheme (Buzacott and Shanthikumar, 1992) uses requisition tags, order tags and PA cards to coordinate and control material and part flow within the manufacturing system (Figure 2.1). A cell is defined as any station in the system that performs some activity involving the material or part. A store is defined as an inventory holding position. Each store holds only one type of product, although several stores may be in the same physical location. A cell/store combination refers to the cell which produces a particular product, and the store to which that product is sent upon completion. If a cell is capable of producing more than one product, then it will be part of more than one cell/store combination.

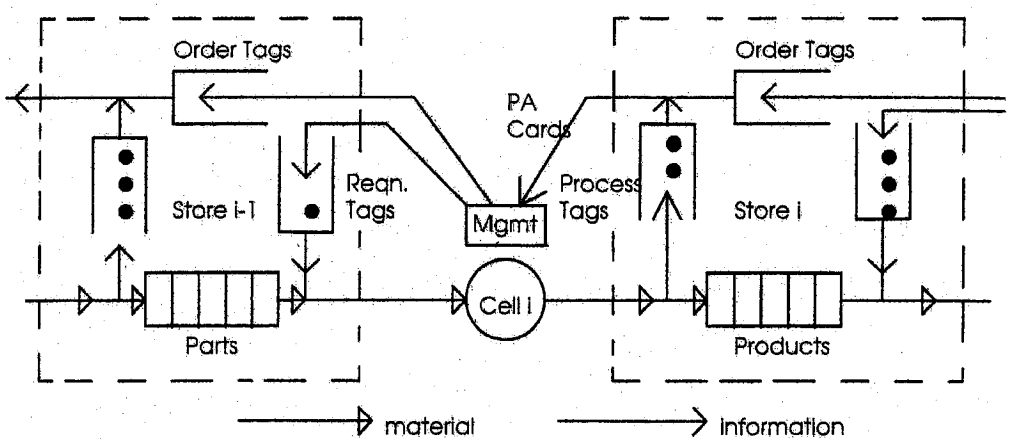

Figure 2.1: PAC Coordination Scheme (Buzacott and Shanthikumar, 1993)

If a cell requires a part or material from an upstream store, it issues a *requisition* tag. If the store has stock, it immediately fills the request. If there is no stock, the requisition tag is held at the store until the demand can be met; therefore, the number of waiting requisition tags represents the backlog at that store. The cell also issues the store an *order* tag; depending on the nature of the control system, this order tag may arrive in advance of the requisition tag, providing the upstream store with advance warning that a part is going to be requested.

At each store, there are a set number of *process* tags. When an order tag arrives from the downstream cell, it is paired with a process tag, if one is available, and this becomes a *production authorization (PA) card.* This PA card is then sent to the cell which produces the part, therefore authorizing that cell to begin production on the part. This cell would then issue order and requisition tags for the required component parts. Once production on the part is complete, the PA card is converted into a process tag, and this tag and the part is sent to the requesting store. If an order tag arrives at a store and there are no process tags available, the tag is held at the store until a process tag becomes available.

The flow of parts and information then is controlled by setting these parameters:

- z_i, the initial inventory at store i ($i = 1,\ldots,m$ where m=number of stores). This will also be the amount of inventory at the store when there are no active PA cards in the system.

- k_i, the number of process tags at each store i
- τ_i, the delay between the issuance of an order tag and the corresponding requisition tag
- r_i, the packet size for transmittal of PA Cards (represents batch production)

The initial inventory parameter, z_i, in fact represents an inventory cap; there can never be more than this amount of inventory in store i at any time. When the system is empty, this value is equal to the amount of inventory in the store. When an order is received for a part, the store immediately begins the process of authorizing the supplying cell to replace the part. Increasing the value of z_i at any cell will increase the probability that an arriving requisition will find stock waiting at the cell, therefore decreasing the probability of a production delay at the requesting cell. Increasing the value of z_i at a store which supplies finished product to customers will result in an increase in the probability of a customer requisition being immediately satisfied, and a decrease in the average customer wait time.

The process tag parameter, k_i, controls the flow of information to upstream cells. When order tags are received at store i, they are paired with a process tag and a PA card is created for the supplying cell. This will also trigger orders for the necessary component parts at upstream stores. If an order arrives at store i and no process tags are available, this means that there are already k_i active PA cards (and therefore a total of k_i jobs in production and in the queue) at the supplying cell. A process tag will become available only when the supplying cell finishes the part currently in production. Until then, the order must wait, and no information about this order is communicated upstream; consequently, no more component parts will be ordered from upstream cells. If the delay at the cell in question was due to a failure, then it is possible that eventually all upstream cells would complete all the orders generated by the PA cards waiting at this cell. All of the upstream stores would have z_i inventory in stock, and further production would be halted. Therefore, the amount of work in process inventory in the system can be controlled by limiting the number of process tags at each store. This will prevent the

"WIP Explosions" (Hopp and Spearman, 2001) which can occur when jobs are released into a system without regard to the current state of the system.

The batching parameter, r, is really only relevant in systems where a single cell may produce multiple products. Batching may be desirable when long changeover times are required at the production cell. If $r > 1$ at a cell/store, then orders for products received by the store from a downstream cell are held there until r orders have been accumulated and matched to process tags; only then is this batch of PA cards created and sent to the cell. If the cell uses a first-in, first-out (FIFO) priority scheme for processing waiting PA cards, then the entire batch will be processed in sequence. It is important to note at this point that there may be any type of scheduling algorithm used by the cell to determine the next part to produce when multiple PA cards are waiting.

The requisition delay parameter, τ, may be used in systems where forecasts provide advance notice of expected demands. Orders for product enable the creation of PA cards to authorize production upstream, while requisitions for product remove stock from the store to be moved to the requesting cell or to the customer. Therefore, with advance notice of upcoming demand, it may be desirable to start production for required component parts in advance of requesting those parts to be delivered. In such a case, a requisition for a part or product will be issued τ time units after the order has been issued.

In order to deal with systems where orders are sent in advance of forecasted demand, cancellation notices may be used to cancel any orders generated for forecasted customer demand which did not materialize. These are propagated through the system, cancelling any as yet unfilled orders for component parts. If any PA cards have already been generated, a decision must be made whether to produce the part or cancel it. Surplus tags are then affixed to any component parts which were produced for the original order. When another order for the part is later received by the store where surplus parts exist, no PA card will be generated at the upstream cell and the surplus part will be used to fill the subsequent requisition. Surplus tags may also be assigned to a part produced as a by-product of another part.

A store may supply more than one downstream cell with the same part; when out of stock, arriving requisition tags are held in a queue and the decision on the sequence in which requests are filled must be made. The priority policy could either be made in advance, as Buzacott and Shanthikumar (1992) assume, or it could be added as a policy parameter, where several common policies could be identified and quantified as inputs to the model. As well, multiple stores may be defined in the model to supply the same part to different cells. In this case, some decision must be made on which store will receive the required requisition tag, and the number of process tags for the part at each store does not have to be the same. This last situation can occur in a production environment where distance is a factor, and therefore a decision is made to have multiple stock points for the same part to reduce travel times to the requesting cells.

2.2 Modeling Traditional Control Strategies using the PAC Scheme

Buzacott and Shanthikumar (1993) have shown that well known shop floor control policies such as Produce-to-Order systems, Base Stock systems, MRP, Kanban, CONWIP, and OPT are simply specialized versions of the PAC scheme described above. For example, Kanban systems can be modeled by setting the number of process tags (k_i) at a supplying store equal to the initial inventory (z_i) at that store. This means that, if a store is empty, all process tags will be in use, and order information from downstream cells will not be passed to upstream cells. It also means that at any time, the number of process tags at the store will be equal to the current inventory at the store. Therefore, if at any time the amount of inventory at the store equals the initial inventory, then all process tags are also at the store, which means no PA cards are currently active. This means that production of the part at the upstream cell will have been halted.

For a CONWIP system, the initial inventory at the final store (z_m) is set to the desired WIP level for the entire system, and initial inventory at all other cells is set to zero. The number of process tags at each cell is set equal to the initial inventory (CONWIP level) at the final store. This is because it is possible for all of the WIP in the system to currently be in process at any cell. There is no delay time between order tags and requisition tags.

For an MRP system, the delay time between the sending of an order tag and a requisition tag (τ_i) is determined by the lead times for those parts or materials. Because this is a push system, there is no limit to the amount of WIP in the system, and therefore all cells have an infinite number of process tags. This means that there is never any delay in the issuing of a PA card upon receipt of an order tag. The initial inventory at each store is defined as the safety stock or planned minimum on-hand inventory. For a discussion on the choice of parameters for other control systems, one can refer to Buzacott and Shanthikumar (1993).

Since the PAC scheme can be used to emulate multiple control strategies simply by changing the parameters discussed above, a single model can be built in order to study the performance measures of the system under these different control strategies. It is also possible to study the performance of a hybrid strategy; for example, a combined 'push-pull' strategy involves the line operating as a pull system from raw materials to a particular cell, and then operating the system downstream from this cell as a push system. This can be achieved by limiting the number of process tags at all upstream stations, thereby controlling the work released into this part of the system, and then eliminating such limits in all stations downstream from the cell. This would make sense in a situation where a bottleneck cell was located somewhere in the middle of the process; limiting production at upstream cells reduces the build up of inventory at the bottleneck, and the bottleneck itself will act as a regulator for the downstream cells, thereby eliminating the need for other control mechanisms at these downstream cells.

2.3 Issues with the PAC Scheme

The time delay parameter, τ, is the delay time between the issuance of an order for a part, and the requisition for this part. In MRP systems, the goal is to produce to a master schedule which is determined from a demand forecast, such that finished products are completed by the time the actual customer demand for that product materializes. Therefore, the time delays, τ_i, at each cell/store should be set to the expected lead time for each product or component up to that stage. By the time the requisition (actual request to

take the component from the store) arrives at the store, the advance order should have triggered the production of a unit to fill that demand in enough lead time to have that product completed and available at the store. Buzacott and Shanthikumar (1993) assume that the forecasts are perfect; otherwise, the system would be required to generate correction or cancellation notices as appropriate. The performance of a manufacturing system where production is scheduled to meet a forecasted demand is affected by the quality of the forecast; therefore, if we assume that the forecasts are perfect, we ignore that effect. This makes it difficult to compare the MRP strategy to alternative strategies that do not depend on such a forecast; one system would receive 'advance warning' of demand while the other receives no such notification and responds to demand as it occurs. If imperfect forecasts were used, the amount of error in the forecast would also make comparison amongst systems difficult. Therefore, we have elected not to include forecasting in the systems we have analyzed for this thesis, and therefore the analysis of MRP type systems is not included. However, since we did not want to preclude the use of this parameter in future work, it will remain a part of the framework, and is also a part of the PACSIM model discussed later in this chapter.

Another issue is with the priority setting schemes. One store may receive requisitions for a single part from more than one downstream production cell. In the case of a store with a backlog of requisitions from more than one downstream cell, the question arises as to which requisition is filled first upon the arrival of a completed part. Buzacott and Shanthikumar (1992) say that when a store receives a finished part from a cell, then the highest priority requisition should be filled first. This implies that the priority setting scheme used is a management decision, and is a characteristic of the system, not one of the assignable parameters within the scheme. Another priority question arises when a single cell is capable of producing more than one part. When the machine is changed from the production of one part to another, there may be setup time involved. The choice of a queuing strategy (FIFO, SPT, longest queue, etc) will affect the performance of the overall system. In our current framework, we have assumed that

the priority setting scheme is a characteristic of the system and is assigned prior to any analysis; in the PACSIM model, we assume that this priority scheme is FIFO.

2.4 Performance Model of a Manufacturing System using the PAC Scheme

Bielunska-Perlikowski (1997) and Bielunska-Perlikowski and Gunn (2002) developed a general performance evaluation model for the PAC scheme using simulation. This model included the ability to study manufacturing systems with complexities such as setup times, travel times for requisitioned material, assembly stations and multiple products. This model has been modified and incorporated into a framework (PACSIM) which permits simulation of multiple design points, and captures several performance estimates for each simulation replication.

2.4.1 Original PAC Simulation Program

The original discrete-event simulation program, *pac.f*, was used as the basis of the new simulation model. It is written in FORTRAN77 and uses the SIMLIB routines of Law and Kelton (2000).

There are seven simulation events (Table 2.1) which may occur in the simulation. The event graph for the simulation is shown in Figure 2.2.

Table 2.1: Event List for the PAC Simulation (Based on Bielunska-Perlikowski, 1997)

Event	Description
1	Arrival of a customer order or part order from a cell
2	Arrival of a requisition at a store
3	Arrival of a Production Authorization card at a cell
4	Part and Process tag arrival at a store
5	Completion of processing of a part at a cell
6	Arrival of work in process inventory at a cell
7	End of the simulation

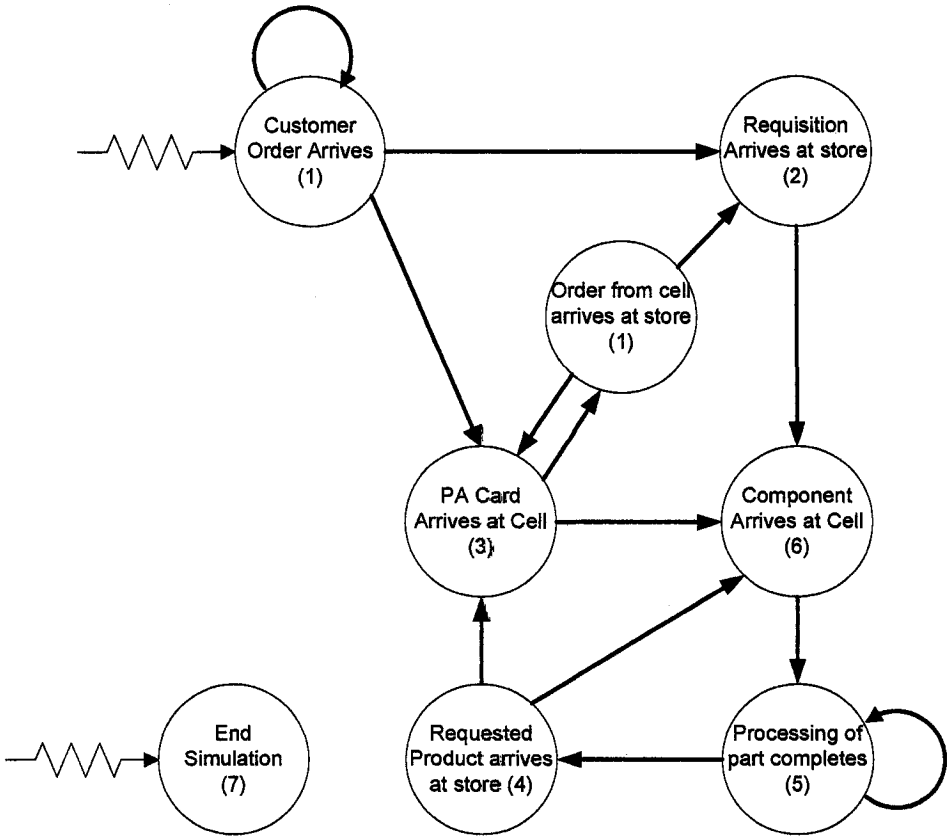

Figure 2.2: Event Graph for the PAC Simulation (Bielunska-Perlikowski, 1997)

The master array of SIMLIB maintains queues and lists. Queues contain entities; each entity is stored as a record with several attributes, including such information as type of product (for product entities or active PA card entities) or time of arrival. At each cell/store combination, there are six individual queues maintained (Table 2.2).

Table 2.2: Queues at Each Cell/Store Maintained by SIMLIB

Queue	Entity Stored	Description
ORD	Orders	Orders awaiting a match with a process tag, or for a batch to be formed
REQ	Requisitions	Requisitions awaiting the arrival of the requested part/product (backlog)
PROD	Products/Parts	All stock at a store which has not yet been requested – entities are distinguished by a product type attribute
PROC	Process Tags	Process tags available at a store for matching with orders
WIP	Products/Parts	Work-in-process inventory – components requested by the cell in response to a PA card, now awaiting processing. May contain several product types.
PAC	PA cards	Active PA cards (awaiting processing or in process)

In addition to the queues, the program maintains several lists which track events and entities (Table 2.3). The main list is the event list, which maintains all upcoming events by event type. The remaining lists are not queues, but rather track the existence of entities in the corresponding queues in Table 2.2 for statistical purposes.

Table 2.3: Lists Maintained by SIMLIB

List	Description
CUST	Waiting customer requisitions (backlog)
CORD	Waiting customer orders
PPRD	Products waiting at each product store – a separate list is maintained for each product type
PWIP	Components waiting to be processed at a cell – a separate list is maintained for each product type
EVENT	The event list

A more in-depth explanation of the simulation model can be found in Appendix A.

2.4.2 Changes to the Original Simulation Program

The queues maintained by SIMLIB contain entities, and problems were encountered for queues which held more than type of entity. Whenever a subroutine had to search the contents of a queue for a particular record, such as when searching a WIP queue for the right type of product, the program would call the appropriate SIMLIB routine to remove the record so that it may be examined. If the record was not the one required, it was added to end of the queue and the next record would be removed. This would be repeated until the right record was found or the entire contents of the queue had been checked. In some cases, when the right record was found, the search stopped. Thus, records within some of the queues were reordered, and this resulted in entities not necessarily being processed in a first-in, first-out manner. The affected subroutines were altered so that whenever a queue was searched, that the entire queue would be put back in the original order. This caused another problem when reporting on the minimum number of entities in a queue; because records were removed in order to be checked, the minimum number of entities in the queue was sometimes reported as one less than the true number. This problem was discovered during verification and was corrected.

Some of the statistical variables used to track performance measures were not being properly reset by the simulation program once the warm-up period was reached. Therefore, some of the performance measures contained observations from the transient period. This was also discovered during the verification experiments, and changes were made to resolve this problem.

One performance measure not originally tracked by the simulation model was cycle time, or the length of time the product remained in the system. Additional statistical variables were added, and the program altered to capture the cycle time whenever a finished product was completed. Where a final product consisted of one or more assembled products, the cycle time for the product was recorded as the length of time the oldest component of the product had spent in production.

Because the simulation was going to be used to provide the data to train neural network metamodels, an additional subroutine, REPORTC, was added to the program to enable the calculation of only the key performance measures, and write those performance measures to individual text files. Another subroutine, SSTATE, was added to track and report on interdeparture times of product from finished goods inventory, as well as other statistics on system performance. These measures would be used later to check when the system had reached steady state during a simulation run.

Other minor changes and additions were:

- New parameters were added to allow for different random number seeds for the calls for generation of new customer orders, the type of product demanded, and the generation of product processing time.
- A new function was added to provide a random observation for a variable with a Weibull distribution.
- The IRANDI function, used to randomly choose the type of product demanded in systems which produced more than one final product, had an improperly defined input variable. Although the arrival of orders was not affected, the type of product demanded was not properly following the given distribution.

- An error in the DEPART subroutine was resulting in a machine being made idle even when parts and PA cards were waiting for processing. This occurred only when the cell was capable of producing more than one type of product.

2.4.3 Overview of the New PAC Simulation Model (PACSIM)

As previously mentioned, the updated simulation program and its associated subroutines formed the core of the new simulation model, PACSIM. Since the purpose of the simulation model was to be able to generate estimates of performance for several different PAC parameter combinations, this original program was changed to a subroutine which is called by the new main calling program, *pacsim.f*.

Information on the system itself, such as the number of cells, products produced at the cells, product routings, average processing times at machines and the average interarrival time of customer demand, are provided to the model in the form of a text file (*pact.in*). The overhead to begin the simulation of a new system is quite low as the computer code does not need to be changed. This file used to be the input to the original program, but is now processed by the new main program. An input file provides the list of PAC parameter combinations (design points) to be simulated. Information on the number of replications per design point, and other run parameters, are contained in another new input file (*today.txt*). After each simulation replication, the model writes the performance estimates, such as average finished goods inventory and customer service percentages, to separate output files. The original *pac.out* report may also be produced, but within the *today.txt* file, the generation of this report can be suppressed (as would be prudent when conducting several simulations). Figure 2.3 shows an overview of the new model. Further details may be found in Appendix A.

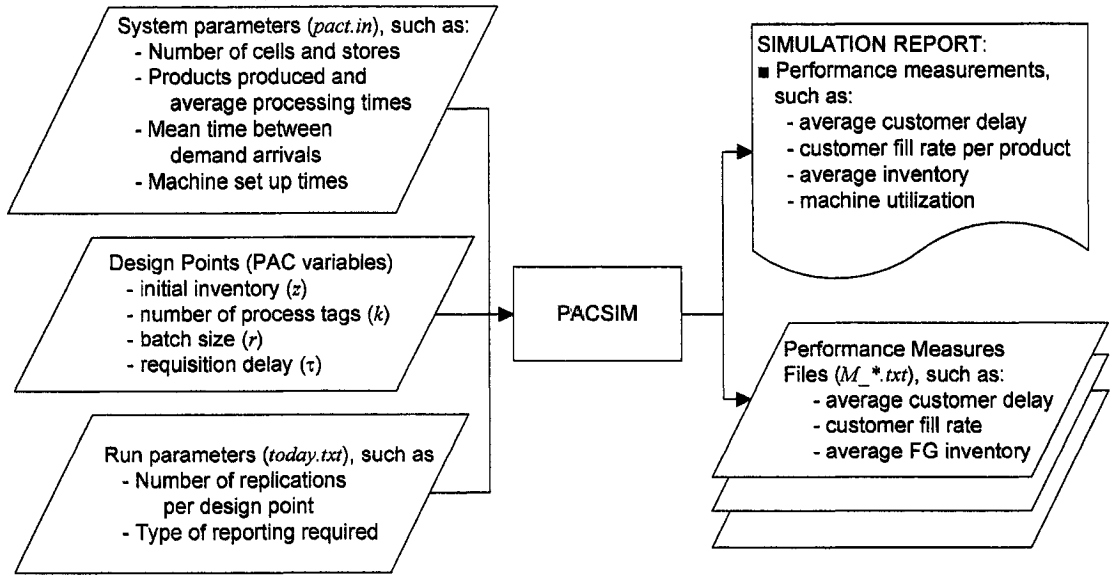

Figure 2.3 Overview of the PACSIM Model

Because it is written in a high level language (FORTRAN), the simulation model is very fast; depending on the complexity of the system, it can execute several hundred runs per minute, which, in our experience, is in the order of ten times faster than simulation packages such as Arena.

2.5 Concluding Remarks

The motivation for this work was the development of a framework to enable analysis of complex manufacturing systems operating under a variety of production control strategies. The PAC scheme, with its ability to model several types of production control strategies, is an attractive modeling scheme. The PACSIM simulation model provides the ability to estimate the performance measures for manufacturing systems with complexities such as multiple products or assembly stations. Chapter 3 discusses the next stage of the framework: analysis and optimization options using the PACSIM model.

CHAPTER 3

ANALYSIS AND OPTIMIZATION OF SIMULATION MODELS

This chapter discusses the options considered for the final component of this framework; the analysis of a manufacturing system operating with the PAC scheme in order to choose the right PAC scheme parameters. In Bielunska-Perlikowski (1997), a simulation optimization approach was used to minimize a cost function, with respect to the PAC parameters, of inventory holding costs and customer delay costs. She acknowledged that, at best, her approach would only result in a local optimum.

We initially explored more sophisticated simulation optimization techniques for the purposes of determining the best combination of PAC parameters for a system. However, it became apparent that a simulation metamodelling approach would provide more flexibility in the analysis of these systems. Some of the factors affecting this decision are discussed in Section 3.1. A discussion of Bielunska-Perlikowski's (1997) approach is provided in Section 3.2. In Section 3.3, we describe several simulation optimization techniques, including the issues with these approaches. Section 3.4 provides an overview of the simulation metamodelling approach, where the goal is to develop a function approximation of the performance measure expected value function. Finally, Section 3.5 summarizes our arguments for the choice of metamodelling over simulation optimization for this framework.

3.1 Important Factors

There are several complicating issues with this problem that make it a challenge. Firstly, three of the PAC parameters (initial inventory, number of process tags, and batch value) are integer values. The number of possible combinations of PAC parameters for a given system depends on the number of processing stations, the number of parts produced, the number of PAC parameters under study at each cell/store, and the range of feasible values for each of these parameters. Even for a relatively small problem, the number of valid combinations can be very large. For example, a four-stage system where only

process tags, k, and initial inventory, z, are decision variables (all $r = 1$ and $\tau = 0$) will have a total of eight parameters. Even when each parameter is limited to one of five possible values, there would be 5^8 or 390,625 possible parameter combinations to analyze.

Although there is some knowledge of the relationship between the PAC parameters and the system performance, such as the expectation that increasing initial inventory at final stores should improve customer service outcomes, the majority of these relationships are not known. Finally, there are several performance measures (outputs) which may be of interest; given the conflicting nature of these measures, any analysis would certainly involve at least two of these performance measures.

3.2 Simulation Optimization of the PAC Performance Model

Bielunska-Perlikowski (1997) was interested in comparing traditional controls strategies to the PAC scheme. She identified eight traditional strategies, including MRP, Local Control, Integral Control, Kanban, CONWIP, Base Stock, and two variants of Produce-to-Order. For MRP, she provided the system with perfect advance notification of demand. She introduced a cost function of customer delay time and inventory levels to the simulation model, which could then estimate the system cost for a given set of input parameters. Her goal was to find, for each of the eight traditional strategies and the PAC scheme, the parameters which minimized this cost function. Then, the best of each strategy could be compared in order to select the best strategy for the system.

She developed a Hooke and Jeeves pattern search algorithm (Hooke and Jeeves, 1961) to find the values of the strategy parameters that optimized the cost function. The procedure involved running a simulation for 300 24-hour days, with a 40 day warm-up period, for each tested point. Because one of the input parameters, τ, was a continuous variable while the others were discrete, Bielunska-Perlikowski limited the range of possible values to be integers, measured in minutes. Bielunska-Perlikowski then proceeded on the assumption that the simulation results were accurate enough that the output of each simulation run could be treated as an exact data value, rather than an estimate, and therefore a deterministic algorithm could be used to optimize the cost

function. In other words, the pattern search approach was applied using the simulation model as the function; each time a function evaluation was required, the simulation model was run at that design point, and the output of the simulation model was used as though it were the exact value of a function.

For the eight traditional strategies tested, the values of the PAC parameters were restricted so that they conformed to the definitions established by Buzacott and Shanthikumar (1992). The optimization technique was applied to the simulation model to determine, for each of these traditional strategies, the values which resulted in the lowest cost system. She then removed these restrictions and again applied the optimization technique to determine the lowest cost PAC scheme. In each case, the PAC scheme resulted in the lowest cost, but since each of the tested strategies could be considered a specialized version of the PAC scheme, this was an expected result.

Bielunska-Perlikowski (1997) also tested a random search optimization algorithm. An initial point was chosen, and K points were randomly generated within a certain range of the initial point. The best point within the test group was determined, which was then designated as the new approximation of the optimal point, and a new set of K random points were generated about this new point, with a slightly reduced range. This process continued until a predetermined number of iterations was reached. She determined that the number of simulations needed to achieve good results was large, and the technique did not perform as well as the pattern search algorithm; however, she concluded that the random search technique could be useful in finding a good starting solution for the pattern search algorithm, or in exploring alternate solutions around the solution provided by the pattern search algorithm.

Bielunska-Perlikowski (1997) did acknowledge that in each case, her simulation optimization technique did not guarantee a global minimum. As well, the technique required the use of a cost function, in order to reduce the multiple outputs of the simulation into a single value to be optimized. In order to determine the total inventory cost, a separate (arbitrary) cost was assigned to the value of WIP at each stage of production, assuming that WIP further downstream in the system was more valuable than

raw material due to the value that had been added by processing. Bielunska-Perlikowski (1997) acknowledged that costing is problematic in that it introduced yet another parameter to the problem, one which may very seriously influence the optimization result. If the original cost coefficients were to be changed, even slightly, the entire optimization procedure would need to be repeated.

While some of these problems could be resolved by applying a more sophisticated simulation optimization technique, we argue that simulation optimization is not the right approach to this problem. In the next section, we provide an overview of several simulation optimization approaches, and the issues with those approaches, in order to support this argument.

3.3 Simulation Optimization

Simulation optimization involves optimizing the parameters of a stochastic complex system by using a simulation model as the objective function (Azadivar, 1999). The optimization problem may be represented by

$$\min_{x \in S} f(x) : f(x) = \lim_{t \to \infty} E\left[L(\xi \mid x,t)\right] \qquad [1]$$

where:
x = the parameter value(s)
S = the range of acceptable values for x
t = the length of the simulation run
ξ = stochastic process which drives the simulation
$L(\xi \mid x,t)$ = random simulation response given x and t

Given that the limit as $t \to \infty$, $f(x)$, is a deterministic function, the optimization problem in equation [1] is well posed. However, we generally will not know how to evaluate f except as a result of two limiting operations: the one as part of the expected value, and the other as $t \to \infty$.

The goal of the optimization technique applied is to find the optimal parameter setting, x^*, such that

$$x^* = \arg\min_{x \in S} f(x) \qquad [2]$$

The choice of optimization approach depends on several factors. The function itself may be multivariate or a single output measure. The parameter(s), x, may be continuous or discrete. There may be system constraints involved. Also, depending on the nature of the problem, it may be acceptable to find a good local solution, rather than attempting to find a global one. This section provides an overview of several simulation optimization approaches, and the issues with the application of these approaches to our problem.

3.3.1 Rigorous Methods

Only two methods, stochastic approximation methods and sample path methods, have been shown to provide a rigorous approach to simulation optimization. Stochastic approximation methods are based on the theory that a local minimum of a function can be found by finding the zero of the gradient (Fu, 1994). The classical stochastic approximation technique (Andradottir, 1998) is a gradient descent technique that involves generating a sequence $\{x_n\}$ of estimates of the solution, x^*, using the equation:

$$x_{n+1} = x_n - a_n Y_n \qquad [3]$$

where

Y_n = an estimate of $\nabla f(x_n)$

a_n = the step size, where $\sum_n a_n = \infty$ and $\lim_{n\to\infty} a_n = 0$

The estimate of the gradient, Y_n, is determined by a gradient estimate technique such as perturbation analysis, likelihood ratios, or finite differences, among others (see L'Ecuyer, 1991, for an overview). These techniques are based on the original work of Robbins and Monro, and Kiefer and Wolfowitz (for a description of these approaches, see Glynn, 1986). However, there have been several problems observed with these techniques, including slow convergence and convergence far away from the optimal solution (Yakowitz et al., 2000), and the absence of a good stopping rule (Kleywegt and Shapiro, 2001). Several variants of stochastic approximation have been proposed to address these issues (for example, Andradottir, 1995b), and while some have been shown

to converge almost surely to the optimal solution, the number of iterations of the algorithms may be extremely large.

The Sample Path (Robinson, 1996) or Stochastic Counterpart Method involves approximating the underlying unknown objective function, *f(x)*, with a deterministic function (Andradottir, 1998). In the context of simulation, this function could be the average of one performance measure for a simulation run of length *t*. It is assumed that:

$$f(x) = \lim_{t \to \infty} f(x,t) \qquad [4]$$

where

x = controllable variable,
t = the length of the simulation run, and
$f(x,t)$ = a function of the output of a simulation.

For a finite t, $f(x,t)$ is dependent on the stochastic components, w, of the simulation. Therefore, this technique involves selecting or generating a fixed w, so that the original function can be approximated by a deterministic function of x, $f(x \mid w,t)$. Therefore, the problem is then to find x^*, such that:

$$x^* = \arg\min f(x \mid w,t) \qquad [5]$$

The assumption is that if t is large enough, this deterministic function will be a good approximation of the true function, *f(x)*, and it may be minimized by applying a deterministic optimization method (Robinson, 1996). The challenging issue that remains is how to ensure that w remains fixed in each simulation run. Robinson (1996) suggests that common random numbers may be used; however, in complex simulation models, this application is not straight-forward, as there may be several stochastic processes involved, and the order in which events occur may matter.

Neither stochastic approximation methods nor sample path methods are commonly applied. Both require very long simulation runs. Stochastic approximation methods are

designed for continuous variables, and the Sample Path Method would be very difficult to implement for complex simulation models.

3.3.1.1 Gradient Estimation Techniques

Gradient Based Search methods estimate the response function gradient and use common mathematical programming techniques to minimize the function (Carson and Maria, 1997). Finite Difference Estimation "is the crudest method of estimating the gradient" (Azadivar, 1999, p.95). The estimate is obtained by simulating a system under a set of input parameters, X, and then for each input in this vector, an estimate of the gradient at X is obtained by

$$\frac{\hat{\delta L}}{\delta x_i} = \frac{[L(x_1,\ldots,x_i+\Delta x_i,\ldots,x_P)-L(x_1,\ldots,x_P)]}{\Delta x_i} \qquad [6]$$

where
P = the number of input parameters, and
L = the random output of the simulation (as an estimate of the function).

This estimate itself is also a random variable. To estimate the partial gradient for all input variables, P+1 simulations are necessary, and in order to get a good estimate of the function, multiple observations may be necessary for each partial derivative. Because L is a noisy estimator of the function, f, it is "quite likely that at least one of the estimates of the gradient point the search in the wrong direction" (Azadivar, 1999, p.95). To be more confident that each estimate within the gradient has the proper sign, several simulation replications would be required, and a confidence interval for the gradient estimate constructed.

Other techniques in this category include Likelihood Ratios, Perturbation Analysis, and the Frequency Domain Method. Once a gradient has been estimated, a mathematical search technique can be used to search for an optimum. These techniques assume that the variables involved are continuous.

3.3.2 Response Surface Methodology

Response Surface Methodology (RSM) is a collection of statistical and mathematical techniques useful for optimizing stochastic functions, not primarily in simulation (Gonda Neddermeijer et al., 2000). In the simulation optimization context, the RSM technique involves first developing an approximation of the simulation objective function by a low order polynomial on a small subregion of the domain considered to be of interest. Then, through successive stages, the size of this optimal search area is reduced by minimizing the low order polynomial and reducing the size of the search area around this point. Then a quadratic or higher order polynomial is fitted and the optimum determined through usual deterministic methods (Fu, 1994).

In general, RSM requires a smaller number of simulation experiments relative to many gradient based methods (Carson and Maria, 1997), however it has been shown that for complex functions with sharp ridges and flat valleys it does not provide good answers (Azadivar, 1992).

3.3.3 Simulation Optimization with Discrete Input Variables

Most of the methods discussed above required that the variables be continuous. As pointed out by Glynn (1986), continuous algorithms and discrete algorithms are very different, and "it is expected that solution methodologies for the discrete problem will need, for the most part, to be tailor-made to the application" (Glynn, 1986, p.54). Simulation optimization techniques for these classes of problems have typically involved the adaptation of heuristic techniques designed for deterministic optimization or the development of specially designed techniques. Some of these are described below.

3.3.3.1 Simulated Annealing

Simulated Annealing (SA) is a technique originally designed for deterministic optimization (e.g. Kirkpatrick et al, 1983) which has been applied to simulation optimization. One of the first applications of SA to simulation optimization was reported by Bulgak and Sanders (1988). At each stage of the algorithm, they conduct two replications of the simulation for both the point under consideration and the current point,

and then calculate a confidence interval for the difference in function values; if there is no statistically significant difference between the average function values, then another replication at both points is conducted and new confidence intervals are calculated. This process continues until a statistically significant difference can be determined, or until the maximum number of iterations is reached. If there is a significant improvement with the new point, it considered a better point, and is adopted; if there is not, then the point may still be adopted based on the magnitude of the average difference and the current temperature.

Haddock and Mittenthal (1992) employ the standard simulation annealing technique to simulation optimization. They attempt to reduce the noise in the simulation responses by conducting very long simulation runs at each point. They present numerical examples which they claim found the optimal solution while only evaluating 30% of the total possible parameter combinations. However, in order to reduce the significant computational time required, they employ a rapid annealing schedule, but this technique is not guaranteed to converge.

Alrefaei and Andradottir (1999) presented a modification of the standard simulated annealing algorithm. They deal with the random response issue by holding the temperature constant, and allowing the algorithm to continue checking points, while recording the number of times each point is visited. In some cases they may even conduct multiple replications at each point at each visit. They also present a second variant of their approach where the optimal solution is the point with the best average performance measure, which is updated every time the point is visited. While the authors show that these techniques converge almost surely to the optimal solution, they require a significant amount of simulation replications. The numerical example presented in Alrefaei and Andradottir (1999) involved one variable and only 50 feasible solutions. In their experiments with these algorithms, 400 – 1000 iterations of the algorithm (sometimes with multiple replications per iteration) were required to converge to the optimal solution.

While some success has been reported with the application of simulated annealing to simulation optimization problems, it takes a significant amount of computational effort at each stage of the algorithm to determine if there is a significant difference between a candidate point and the current solution. The algorithm has been shown not to converge when the estimates of the objective function are noisy (Gong et al., 1999).

3.3.3.2 Tabu Search

Another heuristic originally developed for deterministic optimization which has been applied to simulation optimization is the tabu search algorithm (see Glover et al., 1999). Lutz et al. (1998) used tabu search to find the optimal buffer sizes for a six-station serial production line. At each evaluation stage, four simulation replications are carried out and the average value of the replications is taken as the actual function evaluation, and the algorithm is carried out as in the deterministic case.

Martin et al. (1998) use tabu search to determine the number of Kanbans and lot sizes for a 3 station serial line which produces two products, in order to minimize a single utility function. The value of this function for a given configuration could only be estimated via simulation. At each evaluation stage, four simulation replications were carried out and the average value of the replications, and a 90 % confidence interval for the true mean, were calculated. A move was defined as an improving move only if the move had at least a 90% chance of being indifferent or greater than previous moves. If the simulation response is at all noisy, then most moves will be non-improving under this definition. They tested several variants of tabu search, and found a simulated annealing algorithm found the optimal solution faster than their best tabu search technique.

3.3.3.3 Evolutionary Algorithms

Evolutionary algorithms include various methods such as genetic algorithms, evolution strategies, and evolutionary programming (Pierreval and Paris, 2000). These techniques were also originally developed for deterministic objective functions; however, some researchers have applied the technique to simulation optimization problems (e.g. Pierreval and Tautou, 1997, Tompkins and Azadivar, 1995). While EA's provide the

ability to include qualitative variables, they are often very slow (Pierreval and Paris, 2000). When used for simulation optimization, each time a function evaluation is required, multiple replications of the simulation are conducted and the average of the simulation responses is used as in the deterministic case. One major drawback is the lack of results on the convergence of such an approach, although, as Pierreval and Paris (2000) point out, there have been several positive results reported in the literature.

3.3.3.4 Specially Designed Discrete Simulation Optimization Techniques

There are a few random search algorithms which have been investigated specifically for discrete variable simulation optimization. Andradottir (1992, 1995a) proposed a discrete optimization method where the idea is to generate a random walk through the state space. Starting at a random point, a simulation is conducted and then a neighbouring point is randomly chosen and a simulation is conducted at that point. If the response of the neighbouring point is better than the current point, then the neighbouring point is considered the current alternative. If not, a neighbouring point is again randomly selected and the process repeated. She then shows that the alternative visited most often over several iterations converges almost surely to a local optimizer of the objective function. A similar approach was taken in Andradottir (1996), however instead of limiting the choice of next possible alternative to that of a neighbouring point, any point in the state space may be randomly chosen and compared to the current alternative. Her assumptions include the idea that, even though the simulation generates random responses at each stage of the algorithm, the probability that a correct move is taken is higher than the probability that an incorrect move is taken. This method was shown to converge almost surely to the global solution over a large number of iterations

The stochastic ruler algorithm (Yan and Mukai, 1992) is similar to simulated annealing, however it deals with the stochastic nature of the problem somewhat differently. Using their notation, assume the goal is to minimize the unknown objective function $g(s)$:

$$g(s) = E[h(s, Y(s))] \qquad [7]$$

where

s = the decision variable, such that $s \in S$, the set of possible solutions,
$h(s, y)$ = performance of the system, and
y = a sample of the random vector, $Y(s)$.

A random variable, $H(s)$ is defined by $H(s) = h(s, Y(s))$. Another random variable, $\Theta(a,b)$, represents the stochastic ruler. The values of a and b are chosen, through simulation experiments or experience, such that $a \leq H(s) \leq b \quad \forall s \in S$. In this paper, the authors assign a uniform distribution to Θ, but note that other distributions are possible.

They then turn the original minimization problem into the maximization problem:

$$\max\{P(s,a,b) \mid s \in S\} \qquad [8]$$

where

$$P(s,a,b) = P[H(s) \leq \Theta(a,b)].$$

The solution set for this problem, $S^*(a,b)$, is defined as

$$S^*(a,b) = \{s \in S \mid P(s,a,b) \geq P(s',a,b)\ \forall s' \in S\} \qquad [9]$$

A sequence $\{M_k\}$ of positive integers is chosen such that $M \to \infty$ as $k \to \infty$. The variable X_k denotes the current solution, and the algorithm starts at an initial solution, $X_0 = s_0$. At each stage, k, of the algorithm, a neighbouring point of the current solution, s', is randomly chosen as a candidate solution. A sample of $h(s')$ from $H(s')$ is then drawn (by running a simulation replication). A random observation of θ is then drawn from $\Theta(a,b)$. If $h(s') > \theta$, then the candidate is discarded and the process begins again at $k=k+1$, with $X_{k+1} = X_k$; if it passes this test, then another sample of $h(s')$ and θ is drawn and the test is conducted again. This testing continues until either $h(s') > \theta$, whereby the candidate solution is discarded, or until M_k tests have been conducted, in which case s' is accepted as the current solution ($X_{k+1} = s'$). Therefore, at each stage of the algorithm, the random sample, $h(s')$, must "beat" the sample from the stochastic ruler M_k times if it is to be accepted as the current solution. They showed that the sequence of solutions, $\{X_k\}$, is

a nonstationary Markov chain, and under mild conditions they show that the probability that the current solution, X_k, is the global solution converges to one as $k \rightarrow \infty$.

While Alrefaei and Andradottir (1997) suggested variants to this method to accelerate the convergence, these methods do result in a very large number of simulation replications in order to ensure a global solution. In fact, Yan and Makai (1992) choose their neighbourhood structure so as to ensure that every possible solution in the state space is covered by the algorithm; combining this with the increase in the number of iterations at each stage of the algorithm, it can be easily seen that where the solution space is large, the computational requirements for this algorithm will be considerable. While the Alrefaei and Andradottir (1997) variants involve conducting only a fixed number of simulations at each stage of the algorithm, their method also involves visiting every possible point in the state space at least once. Finally, choosing the stochastic ruler presents another problem; it may involve another process in order to estimate the 'bounds' on the objective function.

The stochastic comparison method (Gong et al., 1999) is similar to the stochastic ruler method in its implementation; however, the test at each stage of the algorithm is simply whether the random sample of the function at the candidate solution is greater than a random sample of the function at the current solution; if the candidate solution passes this test M_k times, it is adopted as the current solution. As well, the candidate solution is not chosen from a neighbourhood of the current solution, but randomly from the entire state space.

3.3.3.5 Comparison Procedures

Methods such as ranking and selection methods and multiple comparison procedures use statistical methods to compare the outcomes of many simulation experiments to determine the 'best' alternative from a discrete set of alternatives (see Swisher and Jacobson, 1999). This approach is useful only when a small number of alternative scenarios are under consideration, and the goal is to determine the best, according to some system measure.

3.3.4 Multiple Response Optimization

There has been some work on simulation optimization for multiple response systems. Evans et al. (1991) detail approaches which involve some type of articulation of preferences from the decision maker. For example, in the manufacturing case, a decision maker may wish to minimize work in process inventory while maximizing throughput and customer service levels. Preference articulation is then having the decision maker decide which outcomes are more 'important'.

Using a cost function, as described above, would be considered a prior articulation of preferences. Fixing a large cost to late orders would result in a solution that keeps customer service levels high, but most likely at the expense of work in process inventory. Also, the problem could be formulated as a goal programming problem, where the decision maker must decide which of the outcomes is most important. Another approach involves a progressive articulation of preferences, where a solution which attempts to find a good compromise amongst the output measures is produced and the decision maker must decide which of the outcomes is least satisfactory and a new solution is determined based on this articulated preference. This routine would be repeated until the decision maker is satisfied with the result. A posterior articulation approach would involve generating a series of (non-dominated) solutions, each of which provides the best possible outcome for only one of the output measures, and allowing the decision maker to choose which is most preferred. Evans et al. (1991) conclude that the best general method for multicriteria optimization would involve all three types of approaches. However, the ability of the decision maker to articulate various types of preference information is perhaps the greatest concern. All approaches can involve running a simulation procedure at least once for every possible outcome, and multiple times in the interactive case.

Mollaghasemi and Evans (1994) approach the problem in an interactive manner for a manufacturing design problem. Initially, the system is optimized with respect to one value at a time, which then determines the 'ideal' value for each measure. Then, an objective function representing the weighted deviation of each output measure from its

corresponding ideal measure is created, and a solution which minimizes this deviation is found using a gradient based approach. The decision maker can review this solution, and then decide if one of the objectives should be improved. The algorithm then puts a higher weight on deviations for this objective and repeats the optimization procedure. This process continues until the decision maker is satisfied with the solution.

Another approach is to determine a minimum (or maximum) constraint on one or more output variables while attempting to optimize another variable (Azadivar, 1999). For example, again in the manufacturing case, a firm may want to minimize work-in-process inventory, but require a certain minimum customer service rate from the system. Azadivar et al. (1996) perform such an optimization on a production system using an algorithm based on the complex search method algorithm.

3.3.5 Simulation Optimization of Manufacturing Systems

Brennan and Rogers (1995) use an infinitesimal perturbation analysis (IPA) technique on a simulation model to determine an operational policy for a production line with machines that fail. The goal was to determine the line's throughput sensitivity to changes in mean time to repair (MTTR) for the stations, in order to identify which stations should receive priority during failures. Sammons and Cochran (1996) use simulation to study the impact of making design changes to a manufacturing work cell. Because there were only three design changes under consideration, and therefore eight possible alternatives, they studied all of the changes and compared them using statistical techniques. However, they did indicate that a pattern search algorithm could eliminate some design alternatives for larger problems.

Bulgak and Sanders (1988) simulated an automatic assembly system, and used a modified simulated annealing algorithm to optimize buffer sizes. Paris and Pierreval (2001) use a distributed evolutionary simulation optimization approach for finding the optimal allocation of Kanbans in a four stage, three product production line. Sengupta et al. (1999) use the Hooke and Jeeves algorithm to find the optimal Kanban allocation in a in a three cell FMS.

3.4 Simulation Metamodeling

An algebraic model of a model is known as a metamodel. A metamodel may be a simplified version of a known formulaic relationship, such as a Taylor-series expansion of a known function (Sargent, 1991), or a function fitted to observations of actual data, usually through linear least squares regression. Simulation metamodels are response surfaces fit to data generated by a simulation model, where the actual functional relationship between inputs and outputs is unknown (Figure 3.1).

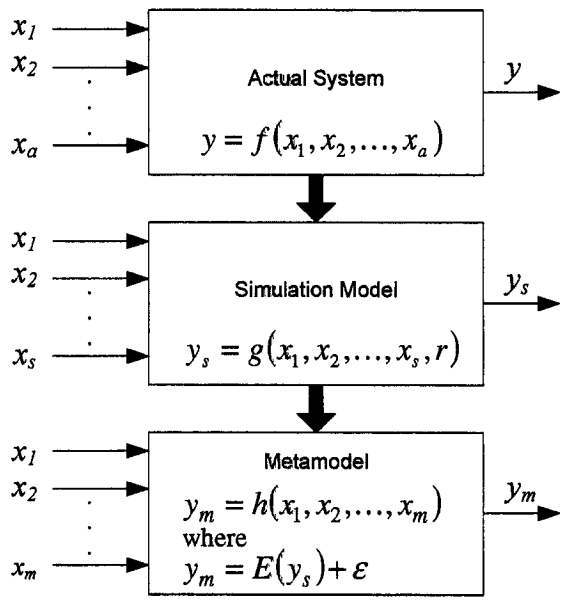

Figure 3.1: Metamodelling Concept

Following a similar definition to that of Kleijnen (1979), the underlying (unknown) function of the actual system, *f*, can be given as

$$y = f(x_1, x_2, \ldots, x_a) \qquad [10]$$

where

x_i = a factor influencing the outputs of the actual system, $i = 1, \ldots, a$.

The simulation model attempts to approximate the output of the actual system

$$y_s = g(x_1, x_2, \ldots, x_s, r) \qquad [11]$$

where
y_s = the output of the simulation,
x_j = a factor chosen to be input to the simulation model, $j = 1...s$, $s \leq a$, and
r = random effects and effects of actual factors not included in the simulation model.

In the case where there are multiple simulation outputs, separate metamodels are typically developed for each output (Barton, 1998, Kleijnen and Sargent, 2000). Finally, a simulation metamodel is an approximation of the simulation model:

$$y_m = h(x_1, x_2, \ldots, x_m) \quad [12]$$

where
y_m = the metamodel estimate of the simulation response, and
x_k = factors chosen to be input to the metamodel, $k = 1...m$, $m \leq s$, and

$$y_m = E(y_s) + \varepsilon \quad [13]$$

where
ε = fitting error, with expected mean of zero.

There are many advantages to metamodelling in post-simulation analysis, as discussed by Friedman and Pressman (1988, p.939):

> "Aside from the obvious advantage that working with a mathematical function has over running and re-running costly simulation programs, and aside from the pleasing elegance of a solution obtained by the union of numerical and analytic techniques, the simulation metamodel has been lauded for its many other uses, among them: model simplification, enhanced exploration and interpretation of the model, generalization to other models of the same type, sensitivity analysis, optimization, answering inverse questions, and providing the researcher with a better understanding of the behaviour of the system under study and the interrelationships among the variables."

While a simulation model can produce very good estimates of the actual measures of interest, the outputs are random variables. Careful design of the simulation experiments can reduce the variance of these measures, but cannot entirely eliminate the noise in the output. If the goal of the analysis is to optimize output measures, then simulation

metamodelling has a distinct advantage over simulation optimization. Metamodels are functional approximations of the simulation expected value functions; deterministic nonlinear optimization techniques may be applied to these functions to find the optimal values.

Major issues in simulation metamodelling include determining the functional form of the metamodel, determining the design points required to fit the metamodel to the simulation model, and determining the adequacy of the fitted metamodel (Barton, 1998).

3.4.1 Forms of Simulation Metamodels

Often, simulation metamodels are linear regression models (Sargent, 1991) of the following form:

$$y = \beta_0 + \sum_{i=1}^{m} \beta_i x_i + \sum_{i=1}^{m} \beta_{ii} x_i^2 + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \beta_{ij} x_i x_j + \cdots + \varepsilon \qquad [14]$$

The procedure for fitting a regression model to simulation output has been extensively reported in the literature (e.g. Kleijnen, 1979, Madu, 1990). Basically, it starts with a low order model with minimal or no interactions, and the complexity of the model is increased until an acceptable fitting error has been achieved. Unlike the Response Surface Methodology approach in simulation optimization (Section 3.3.2), the goal is to fit a regression model to the entire space, rather than a small subsection of the space where an optimal solution to a function is believed to be.

In some cases, there may be some knowledge of the relationship between input and output variables, and a functional form may be chosen based on that knowledge. For example, Friedman and Pressman (1988) developed a simulation metamodel for an M/M/s queuing system, with demands arriving according to a Poisson process with a constant average arrival rate, λ, and s identical servers each with service rate, μ. The measure of interest was the time spent in the system, W. They assumed that this value was dependent upon the utilization of the servers, ρ, where $\rho = \lambda/\mu s$, and therefore chose a multiplicative model:

$$W_i = \frac{\alpha(\lambda_i)^{\beta_1} v_i}{(\mu_i)^{\beta_2}(s_i)^{\beta_3}} \quad [15]$$

where

W_i = wait time observation from the simulation
λ_i = arrival rate for observation i
μ_i = service rate for observation i
s_i = number of identical servers for observation i
α, β_j = coefficients for the metamodel
v_i = error term

They then transformed this function into a linear model by a logarithmic transformation:

$$\ln W_i = \ln \alpha + \beta_1 \ln \lambda_i - \beta_2 \ln \mu_i - \beta_3 \ln s_i + \ln v_i \quad [16]$$

They used regression analysis to estimate the coefficients for the simulation metamodel, then applied antilogs to return to the original predictive metamodel:

$$\hat{W} = \frac{\alpha(\lambda)^{\beta_1}}{(\mu)^{\beta_2}(s)^{\beta_3}} \quad [17]$$

where

$\hat{W}$ = the predicted output of the simulation model.

They conducted 10 experiments where the simulation generated 10 sets of observations. A separate metamodel was constructed for each of these 10 sets of data, and they found that the resulting form of the metamodel was consistent over the 10 experiments. They tested several input combinations with these models and found that the average relative error between the analytical result and the metamodel results varied between 1.36% and 4.71%. The authors used a similar approach to determine the total cost of an inventory system based on ordering policy variables. Other metamodel forms, such as splines, radial basis functions, and spatial correlation models, are described in Barton (1998).

Recently, artificial neural networks have been studied as another approach to simulation metamodelling. They were originally developed as a simplified mathematical version of biological neural networks (Rumelhart et al., 1994). Neural networks have

been applied to several complex problems, such as pattern classification, forecasting, clustering, and function approximation (Jain et al., 1996); for simulation metamodelling purposes, we are interested in networks used to approximate functions. A neural network computes a function (mapping) from input space to output space (Masson and Wang, 1990). This may be viewed as a weighted directed graph in which artificial neurons are nodes and directed edges (with weights) are connections between neuron outputs (dependent variables) and neuron inputs (independent variables) (Jain et al., 1996). Neural networks may be viewed as another form of nonlinear regression analysis, and are particularly attractive for problems where there is little or no knowledge of the relationship between the inputs or outputs.

3.5 Simulation Optimization vs. Simulation Metamodelling for the PAC Model

The purpose of this framework is to enable the analysis necessary to determine the most appropriate control strategy for the manufacturing system in question. The means of performance estimation in this framework involves a simulation model; therefore, the selection of either simulation optimization or metamodelling was dependent upon the type of analysis that would be necessary to determine the appropriate strategy. This section addresses the applicability of these approaches to our problem.

3.5.1 Discrete Parameters in the PAC Model

For each cell/store in the PAC model, there are potentially three discrete variables: initial inventory, z_i, number of process tags, k_i, and the batch size, r_i. Therefore, simulation optimization techniques designed for continuous variables, such as stochastic approximation methods, would not be applicable to this problem.

As for the simulation optimization techniques for discrete variables, as discussed in Section 3.3.3, they could be roughly broken into categories; techniques designed specifically for simulation optimization, and heuristic methods used in deterministic optimization that researchers have attempted to apply to simulation optimization.

Techniques designed for simulation optimization, such as the Stochastic Ruler Method, can be computationally expensive, as they require sampling at a large number of

points, often more than once, due to the fact that the outcome of a simulation run is only one realization of the random variable. In fact, the Stochastic Ruler Method actually ensures that every possible solution candidate is included in the search and evaluated at least once. Given the size of the systems we wish to analyze, this approach is not practical. As for the heuristic techniques, such as tabu search or simulated annealing, they were designed for optimization problems where the objective function can be evaluated precisely, and where a single evaluation is all that is necessary. However, in the simulation environment, we obtain random observations, rather than precise evaluation, of the output measures. In order to create estimates and to reduce the variance of these estimates from the actual (but unknown) values, one could replicate the simulation many times at each point, and use the average as the estimate of performance. However, there is still no guarantee that the level of accuracy of these averages will be sufficient to ensure that the techniques will work well. It is easy to see that these types of techniques can become computationally expensive for even moderately sized problems.

3.5.2 Identification of Costs

There has been little work published in the area of multi-response simulation optimization, perhaps due to the difficulties presented by the stochastic nature of the simulation responses, and the fact that multiple responses often conflict (Tekin and Sabuncuoglu, 2004). Most of the simulation optimization techniques described in Section 3.3 assume that there is only one objective function to evaluate; either there is only one system measure (one output of the simulation) which can be minimized or maximized, or multiple outputs can be combined into a single function to be optimized.

In the manufacturing environment, one cannot evaluate only one measure, because of the interrelationship of these measures. For example, minimizing only work in process inventory may result in low throughput, which would lead to poor customer service. Therefore, a single function that combines the output measures would be required, such as one which applies a carrying cost for inventory and a penalty for late orders. The simulation optimization procedure would then attempt to find the control strategy which would minimize this overall cost. If the cost function were changed in any way, the

entire optimization procedure would need to be repeated. Cost coefficients for such functions in manufacturing are very often difficult to determine (e.g. Silver et al., 1998). For example, it would be difficult to assign a cost to a lost order; while lost profit for a single transaction may be relatively easy to calculate, placing a cost on potentially lost future business would be very difficult to do.

Therefore, to use simulation optimization, it may be necessary to assign somewhat arbitrary costs of the individual measures in the cost function, to assign the level of importance to these measures. For example, if customer service were the most important factor, then a high cost could be placed on customer delays while a lower cost placed on carrying inventory. However, even if the simulation optimization technique were to provide the global optimum for this cost function, there may be parameter combinations which would allow for much lower average inventory while only incurring a small decrease in customer service; management may be interested in such a solution, but it would not be identified in any simulation optimization approach. As well, as discussed earlier, depending on the complexity of the manufacturing system, it could take an extreme amount of computational expense to determine this optimal solution; however, with a deterministic metamodel, a heuristic such as Simulated Annealing could easily be applied to find this solution.

3.5.3 Exchange Curves

An alternate and more useful approach to determining an appropriate strategy is to determine the trade-offs between two or more of the performance measures, and then allow management to make the decision based on this information.

Starr and Miller (1962) introduced the optimal policy curve to aid in the selection of inventory policies for multiple items when the inventory carrying charge and setup costs were not known. Assuming deterministic demand rates, the total cost of an inventory policy, TC_i, for a single item the sum of the setup costs and the sum of the ordering costs:

$$TC_i = A\frac{D_i}{Q_i} + \frac{Q_i}{2} v_i r \qquad [18]$$

where

A = setup (or ordering) cost, \$/setup
D_i = annual demand rate for item i (units/yr)
Q_i = order quantity for item i (units/order)
v_i = unit cost of item i (\$/unit)
r = carrying charge (\$/\$/unit)

The optimal order quantity, Q_i^* is the well known EOQ formula:

$$Q_i^* = \sqrt{\frac{2AD_i}{v_i r}} = \sqrt{\frac{A}{r}}\sqrt{\frac{2D_i}{v_i}} \qquad [19]$$

If the setup cost, A, and carrying charge, r, were known, the optimal ordering quantity for every item could be determined, and the total average value of inventory for the firm, TI, would be

$$TI = \sum_i \frac{Q_i^* v_i}{2} = \sum_i \frac{1}{2}\left(\sqrt{\frac{A}{r}}\sqrt{\frac{2D_i}{v_i}}\right)v_i = \sqrt{\frac{A}{r}}\sum_i \sqrt{\tfrac{1}{2}D_i v_i} \qquad [20]$$

and the total number of setups required per year, N, would be

$$N = \sum_i \frac{D_i}{Q_i^*} = \sum_i D_i\left(\sqrt{\frac{r}{A}}\sqrt{\frac{v_i}{2D_i}}\right) = \sqrt{\frac{r}{A}}\sum_i \sqrt{\tfrac{1}{2}D_i v_i} \qquad [21]$$

The product of these two measures results in a constant term which is independent of the ratio of the cost variables:

$$TI * N = \left(\sum_i \sqrt{\tfrac{1}{2}D_i v_i}\right)^2 \qquad [22]$$

This relationship between the two measures of interest and provides the optimal policy curve. Since this relationship was constructed on the assumption that the inventory policies for each item were the optimal policies, then any combination of total inventory and number of setups on this curve is an optimal choice. The other relationship of interest is the ratio of total inventory to number of orders:

$$\frac{TI}{N} = \frac{\sqrt{\frac{A}{r}}\sum_i \sqrt{\frac{1}{2}D_i v_i}}{\sqrt{\frac{r}{A}}\sum_i \sqrt{\frac{1}{2}D_i v_i}} = \frac{A}{r} \qquad [23]$$

Thus, by selecting a point on the curve as the desired operating point, management implies the ratio of the setup cost to the carrying charge, A/r. This value may then be used to find the order quantities for each individual item. An optimal policy curve is "a most valuable tool in the frequently occurring and difficult cases where satisfactory estimates of the relevant costs are not available." (Starr and Miller, 1962, p. 164)

Brown (1967) assumed that the setup cost would be known, and suggested the carrying charge, r, be considered a "policy variable" as there was "no correct value to use, other than the value that results in what management wants." (Brown, 1967, p. 30). Rather than requiring that management identify the value of this variable in order to determine the optimal policy, instead management selects the policy, which implies a value for r. He termed these optimal policy curves "exchange curves" since they showed how concessions in one performance measure could be exchanged for improvements in the other.

Lenard and Roy (1995) extended this idea to non-deterministic inventory systems. They approximated the "efficient policy surfaces" for two or more performance measures using a series of simulations of the system under different policy variables. The optimal policy surface was then constructed from the performance measures for all non-dominated policies. A policy is considered a non-dominated policy if there were no other policy where all of the performance outcomes were considered better. Figure 3.2 shows how point X is not a non-dominated policy since there is at least one policy (A) where the performance measures are both better than those for point X.

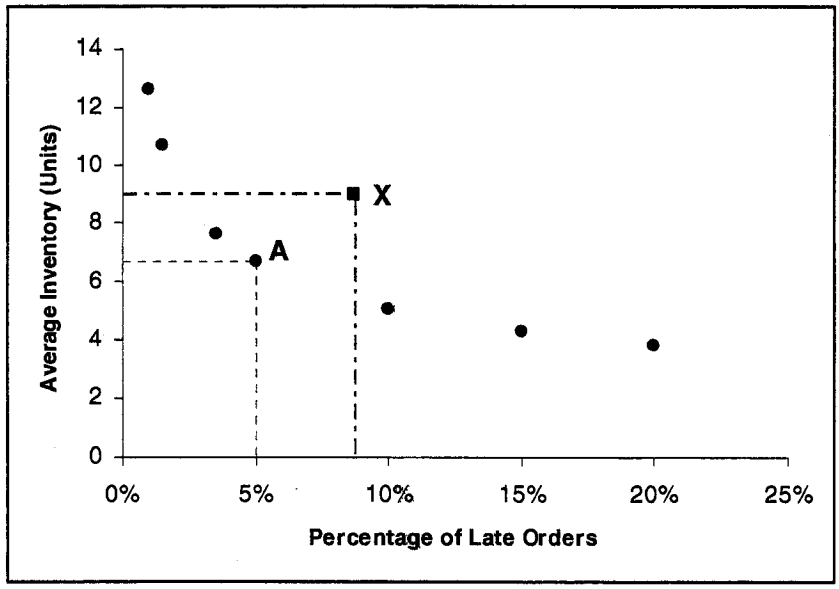

Figure 3.2 Example of Optimal Policy Curve (or Exchange Curve)

When two measures are of interest, this approach is feasible when the measures can be evaluated (or estimated) for all possible policies. The curve itself may be constructed by determining the convex hull of all the performance measure pairs.

When the number of possible policies is large, it may not be possible to obtain all of these pairs. In this case, the curve may be estimated through an optimization approach. Given two conflicting measures, x and y, a cost or score function, C, is given by

$$C = \lambda x + (1-\lambda)y \qquad [24]$$

where

λ = weighting parameter, $0 < \lambda < 1$

By varying the value of λ between 0 and 1, we shift the weight on each of the measures. Repeated minimizations of this function with different values of λ will result in a set of (x,y) pairs which would represent a set of non-dominated pairs. The convex hull of this set represents an exchange curve for this problem.

It is clear from the discussion above that generating such exchange curves would require a great deal of simulation effort compared to the computational effort required if a deterministic function were available to approximate the performance measures.

3.5.4 Optimization with Constraints

Another analysis approach that does not involve identifying costs would be to minimize (or maximize) one of the performance measures subject to constraints on the other measures. For example, minimization of system inventory could be carried out subject to constraints on the customer performance measures. Some discrete simulation optimization methods can deal with constraints, but since each constraint evaluation is also an estimate, it simply increases the complexity of the problem as discussed in Section 3.5.1. However, standard mathematical programming techniques could be applied if metamodels were developed.

3.6 Concluding Remarks

In order to provide an analytical framework for the analysis of a manufacturing system operating under a Production Authorization scheme, simulation metamodels are an essential component. This then raises the question of how to construct these metamodels. Neural networks offer several attractive properties which work for our problem: the ability to model functions where the underlying form of the function is unknown and may be highly nonlinear; and the ability to approximate such unknown functions with reasonable accuracy. In Chapter 4, we will discuss the reasons why neural networks are a highly appropriate choice for this problem.

CHAPTER 4

NEURAL NETWORKS AS SIMULATION METAMODELS

It has become apparent that, in order to fully understand the impact of various PAC combinations on the performance of a manufacturing system, we need to be able to approximate the underlying performance functions for the system. One alternative way of doing this is using neural networks. Any continuous function can be approximated by a feedforward neural network with a single layer of hidden units and continuous sigmoidal transfer functions (Cybenko, 1989). The result is a fully differentiable function, which, as we will see, can provide an unbiased approximation of the expected value function.

The first section of this chapter includes a discussion on the reasons why neural networks were chosen over regression models for this application. Some examples of the application of neural networks in manufacturing are discussed in . The remainder of the chapter discusses in detail feedforward neural networks, the architecture chosen for this framework. For a review of other types of network architectures, see Masson and Wang (1990).

4.1 Neural Networks vs. Regression for Simulation Metamodelling

The question of whether regression or neural networks are the best choice for a metamodelling approach is dependent on the complexity of the problem, and whether or not any *a priori* understanding of the underlying function exists. Regression metamodels are typically constructed in two stages. First, the form of the underlying regression function is selected, and then the parameters of the function must be determined. This is rather straightforward to implement. For data that conforms to a low-order polynomial form, regression has been shown to have significant advantages over neural networks in terms of accuracy (Smith and Mason, 1997). Unfortunately, for manufacturing problems, Yu and Popplewell stated that for modeling manufacturing systems, "multiple, rather

than single, regression is almost always necessary" (1994, p. 789). Neural networks provide a non-parametric approach to metamodelling; no predetermination of the functional form is required. Networks have been shown to be robust to noise (Hurrion and Birgil, 1999) and to deviations from traditional statistical assumptions of normal random errors (Chaveesuk and Smith, 2003). However, the development of neural networks can be difficult. Also, unlike regression models, an examination of the weights of a trained network usually provides no help in understanding or interpreting the system (Hurrion, 1997).

Since the purpose of this framework is to enable the analysis of a complex manufacturing system operating under the PAC scheme, it is not possible to predefine a functional form for the performance measures that is applicable in all cases. As we will see, for even a moderately complex system, second and third order polynomial models are most often not adequate to model these relationships. Although a standard procedure for selecting network size and training does not exist, one can be prescribed for this application.

4.2 Neural Networks as Simulation Metamodels in Manufacturing

While regression analysis has been combined with simulation by several investigators, empirical work on using neural networks as metamodels in discrete event simulation studies is sparse (Savsar and Choueiki, 2000). Savsar and Choueiki (2000) used a simulation-neural network metamodel to determine the optimal number of Kanbans in a just-in-time controlled production line. They compared their neural network metamodel to a regression model, and found that for their problem "the neural network model was superior to the regression model in its ability to interpolate accurately within the design space." (Savsar and Choueiki, 2000, pg. 3262). Hurrion (1997) used a simulation-neural network metamodel to determine the optimal number of Kanbans for a given manufacturing system. In this paper, it was concluded that the use of a simulation optimization technique for this purpose was impractical, because the amount of

simulation time would be excessive for all but the simplest problems, and because the optimal solution found would only be valid for the given cost function.

Kilmer and Smith (1993) compared the use of neural networks and regression models as simulation metamodels. They conclude that, for small numbers of input parameters and output measures, the neural network approach can outperform the first and second order linear regression models typically used in simulation response surface methods. Kilmer et al. (1999) use a neural network metamodel for a simulation of a classical (s,S) inventory problem.

Markham et al. (2000) also compare the use of neural networks, decision trees and regression models as simulation metamodels for finding the optimal number of Kanbans for a production system. They conclude that both the neural networks and decision trees well outperform the regression models.

Mollaghesmi et al. (1998) use a neural network metamodel of a multi-response simulation of a test operation at a manufacturing plant as a decision support system. The desired performance measures of the system are obtained from management (cycle time, WIP, and tester utilization), and the neural network is used to determine the design of the system which can produce those measures (the number of testers at each station and the type of queuing strategy to use). Nasereddin and Mollaghasemi (1999) also use this reverse-simulation metamodel approach, and show that it outperforms a regression metamodel developed from the same set of data.

4.3 Feedforward Neural Networks

The term Feedforward Neural Network (or Backpropagation Network or Perceptron) refers to a network where signals are passed forward through the network from input(s) to final output(s).

4.3.1 Neurons

The basic processing unit in a neural network is the neuron. A neuron is presented with real valued *input* (independent variable) and transforms that input into a single *output* value (or dependent variable). Figure 4.1 represents a simple neuron which

computes the output value, y, for a single input value, x. The neuron first applies the weight, w, to the input value, and then sums this result with the bias weight, w_0. The resulting sum is then transformed via a transfer (or activation) function, f, producing the neuron's output, y.

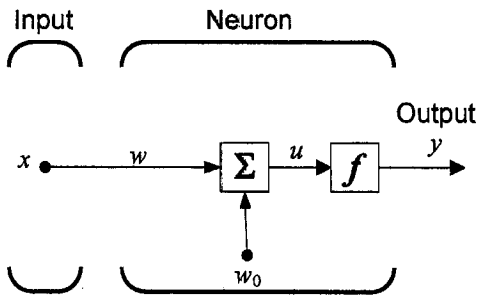

Figure 4.1: A Simple Neuron with One Input (adapted from Hagan et al., 1996)

The resulting output function for this neuron, therefore, is

$$y = f(w_0 + wx) \qquad [25]$$

A neuron may process an input vector, $\mathbf{x}$, with R elements, and in this case, a separate weight is applied to each individual element of the input vector (Figure 4.2).

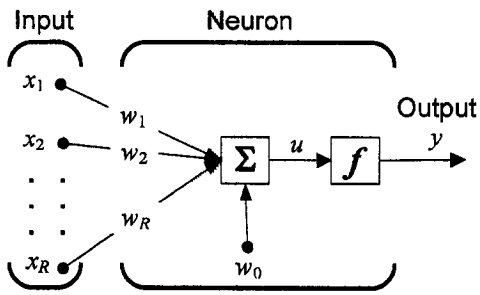

Figure 4.2: A Neuron which Processes an Input Vector (adapted from Hagan et al., 1996)

For this neuron, output function is

$$y = f\left(w_0 + \sum_{i=1}^{R} w_i x_i\right) \qquad [26]$$

There are several different types of activation functions used in neural network neurons. Some of the more common transfer functions are shown in Figure 4.3. The parameter β is an optional slope parameter. The most common transfer function is the logistic function (e.g. Jain et al., 1996).

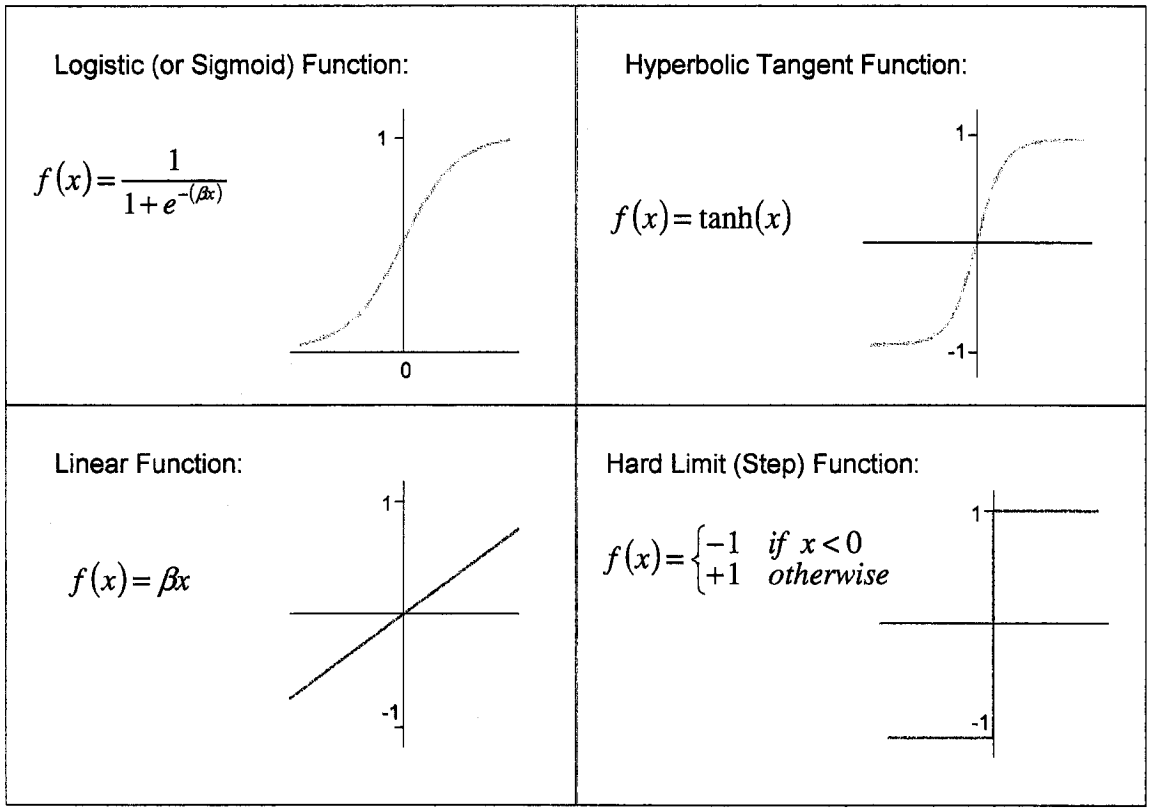

Figure 4.3: Common Transfer Functions

4.3.2 Feedforward Neural Network Architecture

Neurons (often referred to as nodes) in the network are organized in layers, and there may be layers of nodes between the input and output nodes. These are known as hidden layers (Figure 4.4). The transformations at these hidden nodes produce output which is then transmitted as input to the nodes in the next layer.

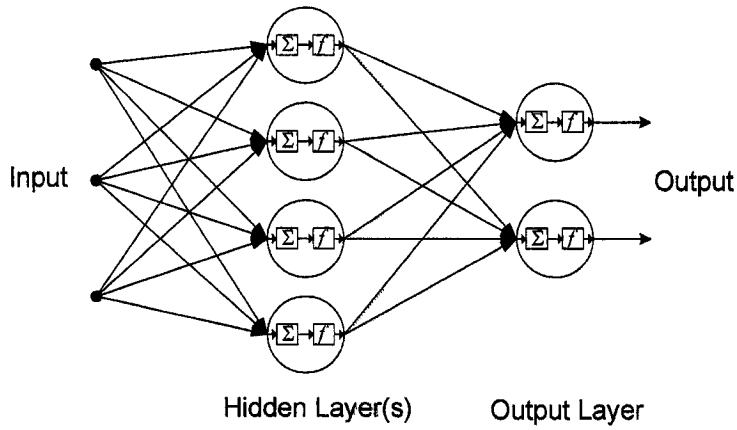

Figure 4.4: A Multilayer Feedforward Neural Network

The details of the network architecture depend on the application for which the network is to be used; the two main categories of applications are function approximation (nonlinear regression) and pattern classification. In pattern classifiers, multiple output nodes are required, and the activation function used in the output layer is nonlinear. For function approximation, the output layer neurons are linear (Haykin, 1994), and there may only be one output node in the layer. Constructing a separate neural network metamodel for each performance function is a common approach in simulation metamodelling (Barton, 1998, Kleijnen and Sargent, 2000). Where multiple output nodes are used, they "can interfere with the learning of each" (Kilmer et al., 1999), and therefore constructing separate networks alleviates this problem. The remainder of this discussion is focused on feed-forward networks with a single output node used for function approximation.

The notation used from this point on follows that of Smith (1993). For a feedforward neural network with a single hidden layer, there are I input nodes (the number of independent variables), and J hidden nodes. The single output node is the dependent variable (Figure 4.5).

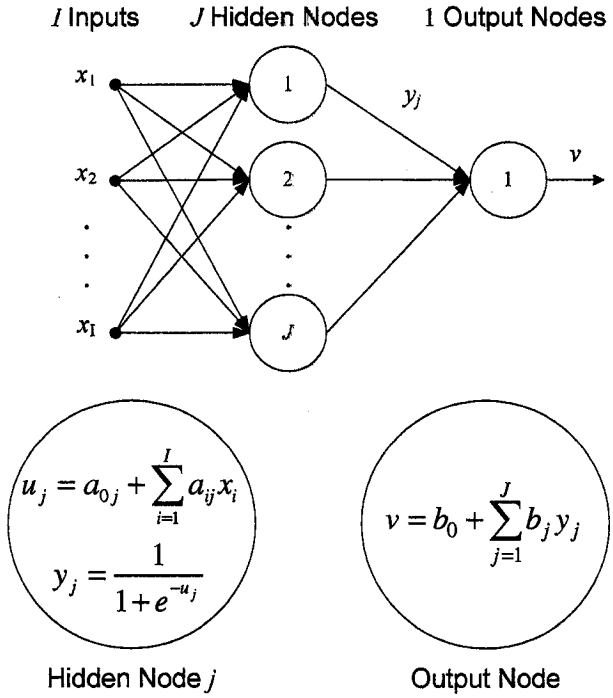

Figure 4.5: A Feedforward Neural Network with a Single Hidden Layer and One Output Node

At each hidden node in this network, the weighted sum of inputs, u_j, is computed as follows:

$$u_j = a_{0j} + \sum_{i=1}^{I} a_{ij} x_i \quad \forall j = 1, \ldots, J \qquad [27]$$

where

a_{0j} = the bias weight for the j^{th} hidden node
a_{ij} = the network weight applied to the i^{th} input to the j^{th} hidden node
x_i = the i^{th} component of the input vector, **x**.

This weighted sum is then transformed by the transfer function (in this case the logistic function) to determine the output of the j^{th} hidden node, y_j:

$$y_j = \frac{1}{1+e^{-u_j}} \quad \forall j = 1, \ldots, J \qquad [28]$$

The output of the hidden layer, y_j, are then passed to the output node. Here, the final output of the network is determined. In this case, a linear transfer function (with slope β = 1) is used at the output node, and therefore the output is simply the weighted sum of the input from the hidden nodes:

$$v = b_0 + \sum_J b_j y_j \qquad [29]$$

where

v = output of the output node
b_{0k} = the bias weight applied at the output node
b_{jk} = the network weight applied to the j^{th} hidden node output at the output node

The resulting function [30] for each network output, v, is a nonlinear function in **x**.

$$v = b_0 + \sum_{j=1}^{J} b_j \left(1 + e^{-\left(a_{0j} + \sum_{i=1}^{I} a_{ij} x_i \right)} \right)^{-1} \qquad [30]$$

The network described above is capable, after assigning values to the weight variables, of producing real-valued output in response to an input vector, **x**.

4.4 Neural Network Training

For function approximation, the network weights which provide the best input-output mapping must be determined. In other words, the model must be fitted to the data. In the neural network literature, this process is called network training or network learning. The goal is to determine the network weights which minimize the difference between the output of the network, v_k, and the corresponding actual or estimated system response, t_k.

A *training dataset* is comprised of input/output *example pairs*. Each example pair consists of a set of input data (independent variables) and the corresponding observed or actual response (dependent variables) obtained through experiment or observation. During *supervised* training, an input/output pair is presented to the network, and a learning algorithm adjusts the weights to reproduce these outputs with a tolerable error.

Typically, the problem is formulated as a least squares problem, although other functions, such as a logarithmic error function (Matsuoka and Yi, 1991), have been used. The example pairs may be presented one at a time (online learning), and the learning algorithm applied to adjust the weights, or the entire training dataset may be presented at each iteration of the algorithm (offline or batch learning). Online learning is appropriate when all of the data is not available at the start of training, or if the function may change over time. In this application, batch learning is used.

In this application, we assume that there are N example pairs, $\boldsymbol{X}_n$,$\boldsymbol{T}_n$, with inputs $\boldsymbol{X}_n = \{x_{1n}, x_{2n}, \ldots, x_{In}\}$ and observed simulation outputs $\boldsymbol{T}_n = t_n$, for all $n = 1, \ldots, N$. When presented with the input, $\boldsymbol{X}_n$, the network will produce output v_n. The goal of the training will then be to minimize the mean squared error function:

$$E = \frac{1}{2N} \sum_{N} (v_n - t_n)^2 \qquad [31]$$

where

v_n = the network output for the n^{th} example, and
t_n = the actual or observed output for the n^{th} example.

Some approaches to network training are discussed below.

4.4.1 Backpropagation (BP) Training

Backpropagation (Rumelhart et al., 1986) is a popular neural network learning algorithm because it is a relatively straight-forward technique which often produces good results (LeCun et al, 1998). Multi-layer perceptrons (networks with one or more hidden layers) trained with backpropagation "have been found to perform well in many applications and to find good solutions to the problems posed" (Lippman, 1988, p.4).

Backpropagation (BP) is a gradient descent optimization procedure that adjusts weights to reduce the system error (Patterson, 1996). The BP method can be used for supervised training, where the transfer functions are differentiable. It also requires that the error function itself is differentiable, a requirement met by the use of the mean squared error (MSE) function (Gallant, 1993). During training, the example sets are

presented to the network and the resulting outputs are calculated (a feedforward pass). Then, the partial gradient of the error functions with respect to the network weights are fed back through the network (backward pass) in order to correct the weights.

The partial gradients of the error term [31], with respect to the network weights, are then given by the following (for the derivation of these formulas, see Appendix B):

$$\frac{\partial E}{\partial a_{ij}} = \sum_{N} \left[(v_n - t_n) b_j \right] y_{jn} (1 - y_{jn}) x_{in} \qquad [32]$$

$$\frac{\partial E}{\partial b_j} = \sum_{N} (v_n - t_n) y_{jn} \qquad [33]$$

At a stage during the algorithm, m, the weights for the next stage, $m+1$, are determined by

$$a_{ij}^{m+1} = a_{ij}^{m} - \eta \frac{\partial E}{a_{ij}^{m}} \qquad [34]$$

$$b_j^{m+1} = b_j^{m} - \eta \frac{\partial E}{b_j^{m}} \qquad [35]$$

where

η = the learning rate (step size)

The learning rate, η, for each change is initialized prior to training, and is often adapted during the training. The procedure is started by initializing the weights randomly with very small values. A first pass through the network is calculated, and then the derivatives computed for each network weight. The weights are updated during the backward pass, and the output values, v_n, are recalculated. The procedure is then repeated until the error term becomes sufficiently small.

4.4.1.1 Modifications to Backpropagation

The original backpropagation learning technique is a gradient descent procedure, and is therefore subject to the problems of any descent procedure – namely, the problem of local minima (Rumelhart et al., 1986). As well, depending on the complexity of the network, backpropagation can take a very long time to converge. Therefore, many researchers have worked on improving this algorithm to reduce the necessary training time.

Rumelhart et al. (1986) proposed a momentum term to help speed up learning. Since a large learning rate can lead to oscillations, the momentum term includes information on past weight changes, which influence the direction the weight will change at each step. For example, for the hidden layer weights at the m^{th} stage of training, the update formula would be:

$$a_{ij}^{m+1} = a_{ij}^{m} - \eta \frac{\partial E}{a_{ij}^{m}} + \alpha\left(a_{ij}^{m} - a_{ij}^{m-1}\right) \qquad [36]$$

where

α = the momentum constant, and $\alpha \in (0,1)$.

Adaptive Learning Rates (see Smith, 1993) uses separate learning rates for each weight in the network, and these rates are changed according to the direction in which the error has been decreasing recently. Other attempts to increase convergence of this technique include normalizing the inputs and/or output values about zero, different ways of initializing the weights prior to training, and other means of automatically adjusting the learning rates (see LeCun et al., 1998).

Another common problem during training is known as weight saturation or network paralysis. When a sigmoid function is used as the transfer function, as is the case here for y_j (equation [28]), problems result as the absolute value of the input to this function, u_j, becomes larger (e.g. Rohwer, 1991), resulting in y_j approaching either 0 or 1. This is because the derivative of the network output with respect to the input layer weights (equation [32]) contains the term $y_j\,(1 - y_j)$, which approaches zero in either case. When this occurs, further learning is essentially halted. This can occur in the early stages of training, when the weights are far from optimal and the gradients of the error function are relatively large, resulting in large weight changes. This has led to researchers studying techniques such as weight decay (eg. Bos and Chng, 1996), bounded weights (Stinchcombe and White, 1990) and penalty terms (e.g. Saito and Nakano, 2000) to prevent weights from becoming too large during network training.

4.4.2 Other Approaches to Neural Network Training

Neural network training has been the subject of a great deal of literature since 1986. This research could be broken into roughly two categories: ad-hoc techniques, such as varying the learning rate, using momentum and rescaling variables; and standard numerical optimization techniques (Hagan and Menhaj, 1994).

The Newton method, in its original form, has been generally felt to be unsuitable as a solution method for this problem; firstly, because of the need to compute and invert the Hessian matrix at each iteration, and secondly, because the Hessian is generally not positive definite everywhere (LeCun et al., 1998) and can be ill-conditioned (Saarinen et al., 1993). Equation [37] shows the Newton update formula for the weights in the training problem.

$$\mathbf{w}^{m+1} = \mathbf{w}^{m} - \left(\nabla^2 E\right)^{-1} \nabla E \qquad [37]$$

where

$\mathbf{w}$ = the vector of network weights.

Modifications to the Newton method for neural network training have included line search methods, such as Armijo's Rule (see, for example, Bertsekas, 1999), which ensure sufficient progress at each iteration (Battitti, 1992). A modified Cholesky factorization (Dennis and Schnabel, 1983) can be used when the Hessian is not positive definite (Battitti, 1992).

The Conjugate Gradient Method has been applied to network learning in cases where the number of weights, and therefore the size of the Hessian matrix, makes calculating the inverse of the Hessian (or an approximation of the Hessian) computationally prohibitive (e.g. Johansson et al.,1992). This is common in such applications as image processing or character recognition, where the number of inputs can be quite large. For such large scale networks, the Conjugate Gradient Method is generally felt to be a good technique for network learning (Moller, 1993). The use of the momentum term discussed earlier can be considered as an approximated form of the conjugate gradient method (Battiti, 1992). Secant methods such as the Broyden, Fletcher, Goldfard and Shanno Update (BFGS) provide a positive definite approximation to the Hessian matrix, using

only information about the gradient. Another more recent approach to the network learning problem is that of Interior Point methods (Trafalis, 1999). The Primal-Dual algorithm (Trafalis, 1999) formulates the training problem as a non-convex nonlinear optimization problem with linear constraints. These constraints on the weights are chosen as to avoid saturation of the neuron outputs.

There are special methods which have been developed for minimizing nonlinear least squares problems, such as the error function in the network training problem. The Gauss-Newton method is based on simplifying the computation of second derivatives and the Levenberg-Marquardt method (Levenberg, 1944, Marquardt, 1963) is a modification of the Gauss-Newton method (Battiti, 1992). We have chosen to use the Levenberg-Marquardt method for this problem; an overview of this method is presented in the next section.

4.4.3 The Levenberg-Marquardt Method

The Gauss-Newton Method (see Appendix C for more details) involves approximating the Hessian matrix using the using the Jacobian matrix, **J**. To simplify the notation, assume that the weights may be represented by a vector $\mathbf{w} = \{a_{01}, \ldots, a_{IJ}, b_{01}, \ldots, b_{JK}\}^T$, where w_j represents the j^{th} weight in this vector, and the vector has M terms. The goal is to minimize the MSE function, E (equation [31]), with respect to the weights, **w**. If the network has one output variable, and there are N example pairs in the training set, then the error function, E, will have N terms. Therefore,

$$e_n = \tfrac{1}{2}(v_n - t_n)^2 \quad n = 1,\ldots, N \qquad [38]$$

where

e_i = the i^{th} error term in the MSE function, and

$$E(\mathbf{w}) = \tfrac{1}{2}\sum_{n=1}^{N} e_n^2 = \tfrac{1}{2}\mathbf{e}^T\mathbf{e} \qquad [39]$$

where

$\mathbf{e} = [e_1\ e_2\ \ldots\ e_N]^T$.

The Jacobian matrix, **J**, is a matrix which includes the derivatives of each error term with respect to the weights, w_j. The entries in this matrix are defined as

$$J(i,j)=\frac{\partial e_i}{\partial w_j} \quad [40]$$

The gradient of the error term with respect to the weights may then be written as

$$\nabla E(w)=\mathbf{J}^T\mathbf{e} \quad [41]$$

Furthermore, the Hessian matrix can be written as:

$$\nabla^2 E(\mathbf{w})=\mathbf{J}^T\mathbf{J}+\mathbf{S} \quad [42]$$

where

$$\mathbf{S}=\sum_{n=1}^{N} e_n \cdot \frac{\partial^2 e_n}{\partial w_k \partial w_j} \quad [43]$$

If **S** is assumed to be small, then Hessian may be approximated by

$$\nabla^2 E(\mathbf{w}) \approx \mathbf{J}^T\mathbf{J} \quad [44]$$

This approximation then replaces the Hessian in the Newton update equation:

$$\mathbf{w}_{s+1}=\mathbf{w}_s-\left(\mathbf{J}^T(\mathbf{w}_s)\mathbf{J}(\mathbf{w}_s)\right)^{-1}\mathbf{J}^T(\mathbf{w}_s)\mathbf{e}(\mathbf{w}_s) \quad [45]$$

It is possible that the approximation may not be positive definite, and therefore the matrix would be invertible. A modification of this method is to augment the approximation with a multiple of the identity matrix:

$$\mathbf{G}=\mathbf{J}^T\mathbf{J}+\mu\mathbf{I} \quad [46]$$

Then, the update equation becomes

$$\mathbf{w}_{s+1}=\mathbf{w}_s-\left(\mathbf{J}^T(\mathbf{w}_s)\mathbf{J}(\mathbf{w}_s)+\mu\mathbf{I}\right)^{-1}\mathbf{J}^T(\mathbf{w}_s)\mathbf{e}(\mathbf{w}_s) \quad [47]$$

This is known as the Levenberg-Marquardt method (Levenberg, 1944, Marquardt, 1963). (See Appendix C for a full description). This method has been found to be very efficient in training networks with up to a few hundred weights (Hagan and Menhaj, 1994).

4.5 Issues in Choosing a Network Architecture and Training

Generalization is the term used to describe the ability of a trained neural network to fairly accurately predict the output for a given input not included in the training dataset (Haykin, 1994). Essentially it is the ability of the network to interpolate. This ability is usually influenced by the size of the training dataset, the samples within the dataset, the architecture of the network (mostly the number of hidden nodes and layers), and the complexity of the underlying function to be mapped. In order to provide "good" generalization, much literature on this subject centres around the number of nodes to include in the network, the number of example pairs to include in the training set, and when (and how) to determine when the network has been sufficiently trained.

The ability of a neural network to learn a complex mapping grows as the number of hidden nodes is increased (e.g. Bebis and Georgiopoulos, 1994). However, it also increases the possibility that the network is *overparameterized*, and will memorize or *overfit* the data (e.g. Chaveesuk and Smith, 2003), leading to poor generalization. This is of special concern when the data contains noise (as is the case with simulation data) and the training sample is small. If the training data is limited, this can be a serious concern; however, this situation may be avoided by simply generating more training points (e.g. Smith, 1993, Demuth and Beale, 2001). Not everyone agrees on how many more, but Haykin (1994, p.183) wrote "as a rule of thumb, we may state for good generalization the number of training examples should be desirably larger than the ratio of the total number of free parameters in the network to the mean squared estimation error."

Another phenomenon widely reported in the neural network literature is that of *overtraining*. Theoretically, this can occur even when the training algorithm is continued too long, and the network begins to learn the data in the training set too closely; therefore, this leads to poor generalization. Again, this is a concern when the training dataset size is limited. Some techniques used to prevent this from occurring include early stopping, and cross-validation (e.g. Twomey and Smith, 1998), where the training is stopped when the mean squared error of the network estimates compared to a validation test set starts to

increase. Since we have the ability to generate many more training examples than we have network weights, such techniques will not be used in this application.

While there have been no definitive rules for choosing the right number of hidden nodes to learn the mapping, one commonly held opinion is that the networks should be trained with as few hidden nodes as necessary to adequately learn the mapping (Masson and Wang, 1990). One approach to selecting the number of nodes is known as the Constructive Approach (Kwok and Yeung, 1997), which starts with a network with a small (maybe one) hidden node, and nodes are added until a network can be trained to an acceptable level of error. Network Pruning methods, such as Optimal Brain Surgeon (Hassibi and Stork, 1992), is the opposite approach: a network is trained with more weights than estimated to be necessary, and then weights are systematically deleted until no more weights can be eliminated without a large increase in the MSE.

4.6 Concluding Remarks

Feed-forward neural networks compose the final element of the framework for analyzing performance of manufacturing systems operating under the PAC scheme. In the next chapter, we will discuss the framework in detail, including the construction of the feed-forward neural networks for our problem, and their use in performance analysis.

CHAPTER 5

A FRAMEWORK FOR MODELING AND ANALYSIS OF PRODUCTION CONTROL SCHEMES

As previously discussed, there are a variety of strategies, such as Kanban or CONWIP, which are used to control the production of a given manufacturing system. Each of these control strategies has a variety of parameters to be set (such as number of Kanban cards per station or CONWIP levels). Therefore, the decision maker must determine which strategy to apply, and determine the appropriate parameter settings for that strategy.

This chapter presents a framework for modeling and analyzing this problem for a given manufacturing system. The PAC coordination scheme (Buzacott and Shanthikumar, 1992), the general performance simulation model (PACSIM), and neural network simulation metamodels form the essential elements of this framework. Using this approach, the decision maker can determine the optimal parameter settings for a control strategy which optimizes some combination of the performance measurements; however, and perhaps more importantly, this framework will also provide the decision maker with the ability to analyze the effects of the parameters on the system.

An overview of the framework is shown in Figure 5.1. The first step, described in Section 5.1, is to initialize the PACSIM model by entering system characteristics, such as process times and demand arrival information, as well as choosing which PAC parameters are assignable variables, and which performance measures will be analysed. Section 5.2 describes the second step, which involves determining the ranges for the input values (PAC parameters), through experimentation and through the application of rules for the choice of parameters within combinations to ensure only feasible combinations are chosen. Section 5.3 discusses the sampling procedure for selecting the design parameter combinations for the training dataset (Step 3). Step 4 involves the selection of simulation run parameters, such as run length and warm-up period, in order to generate steady-state observations of system performance, and is explained in Section

5.4. Step 5 is the construction of the neural network metamodels using the training dataset generated in Steps 3 and 4, and is described in Section 5.5. Finally, Section 5.6 provides a discussion on the various options for analysis using the constructed metamodels, which is the final step.

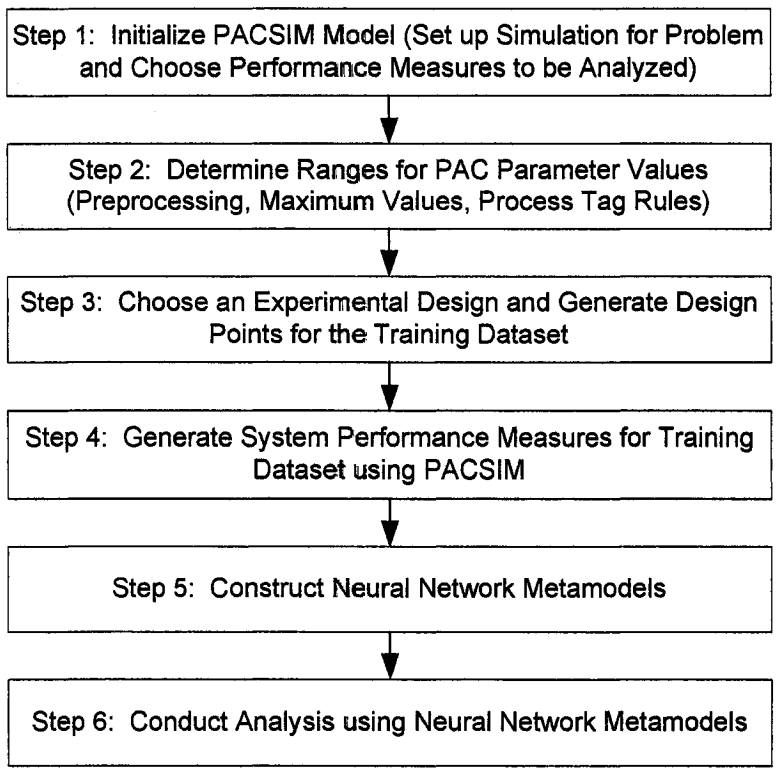

Figure 5.1: Framework for Modeling and Analysis using the PAC Scheme

5.1 Initializing the PACSIM Model

The primary PAC scheme parameters of initial inventory, z, number of process tags, k, and, where relevant, the batch processing parameter, r, are used as control strategy parameters for each cell/store combination in the manufacturing system. The delay parameter, τ, may be dealt with in several ways.

5.1.1 Delay Parameters and Priority Setting Policies

As discussed in Section 2.3, we have chosen to assume that the delay parameter, τ, is not used the models we will analyze; however, this does not exclude the use of this parameter within this framework. This parameter has been included in the PACSIM model, but it is set to zero for all of our experiments. If one were interested in studying and comparing only MRP type systems, then, through the use of a forecast (perfect or otherwise), these systems could be studied using this framework.

For the priority setting issue, we will assume that all stores and cells follow a first in, first out (FIFO) priority policy for PA card and requisition tag queues. As previously mentioned, this will affect the performance of the system, especially in the case of cells which produce multiple parts where setups are required. An opportunity for future work involves the integration of the type of priority strategy used at a cell/store for production or for satisfying requisitions as another assignable system parameter.

5.1.2 Selection of Output Measurements

The performance of a manufacturing system cannot be summarized by a single measurement. As previously discussed, there are often several conflicting performance measurements which should be captured and analyzed in order to understand the "goodness" of a control strategy. The following are measures that we have identified as common system performance measurements, all of which are provided by the PACSIM model:

- **Customer delay time**: For all customers whose demand is not immediately filled from inventory, this measure represents the average time the customer must wait. In the case of a produce to order system, this represents the order lead time.
- **Cycle Time:** Time from the moment raw material is brought into the system until the product arrives at finished goods inventory. In the case of assembly, the "oldest" component determines the cycle time of the product. When there

are no assembly operations, this measurement can be used to estimate work in process inventory (Little's Law).

- **Customer Fill Rate:** Percentage of orders filled immediately from finished goods inventory.
- **Number of Backlogged Customer Requisitions:** The average number of customer requisitions waiting to be filled.
- **Finished Goods Inventory:** Average number of finished products in inventory
- **Work in Process Inventory:** The average inventory of each raw material or partially completed part in the system that is either waiting for processing at a cell or in a store waiting to be requisitioned. This measure may or may not include parts in process at a cell.
- **Throughput:** The average production volume per unit time, per product.

The conflicts between these performance measures are generally well understood (e.g. Hopp and Spearman, 2001), although the magnitude of the impact depends on the system under study. For example, it is usually desirable to maintain low inventory, but this will have an impact on a system's ability to respond to random demand; thus, lower inventory may result in higher customer delay times, lower customer fill rates and a higher average backlog of customer orders. Increasing the average work in process inventory may result in higher throughput, but will also result in longer cycle times.

The measurements to be analyzed depend on the nature of the manufacturing system under study. In the example systems we analyze in Chapter 6, we assume that the goal of the system is to simply keep pace with randomly arriving customer demand; in other words, the throughput must be equal to the customer demand, and therefore throughput is not an interesting measure. However, if it were assumed that there was sufficient demand for all products a system were capable of producing, then throughput would be an interesting measure and should be studied.

The number of metamodels to be developed with the training data can be reduced by reducing the number of performance measures (outputs) selected for analysis, and

therefore requiring fewer networks. This is accomplished by selecting only those measurements that would of interest in the subsequent analysis. For example, if the manufacturing system being studied was a system where products were customized at the final station, then no finished goods inventory could be held. Therefore, fill rate and average finished goods inventory would not be relevant in our analysis. In fact, this would also eliminate the input parameter z for the finished goods store from the metamodel (as it would be fixed at zero), thereby reducing the number of input nodes. However, for the purposes of this work, we will assume no such limitations; our goal is to develop a metamodel which encompasses many possible configurations. At the analysis stage, however, such limitations or fixed values could easily be applied.

5.2 Establishing Ranges of Valid Parameter Combinations for the Training Dataset

Once the parameters for inputs are chosen, a decision must be made as to the range of values from which to select values for each parameter in each combination (the input space). Because the neural networks will be trained using this dataset, they will only be valid for the ranges specified for each parameter in the training set. The minimum parameter values must be determined in order to ensure the system can respond to the customer demand. Maximum values should also be chosen, based on the nature of the system. Finally, parameter combinations must be chosen so as to ensure feasible systems; therefore, the rules used to produce the training set must be known when using the neural network for analysis.

5.2.1 Determining Minimum Feasible Parameter Values

First, we must ensure that the parameter combinations we will use are feasible for the system under study. By feasible, we mean that the system must be capable of achieving a throughput at least equal to the customer demand rate given the PAC parameters. The throughput is directly related to the amount of work-in-process inventory in the system; this WIP is controlled by limiting the flow of information through the system. If this information flow is constrained too much, the throughput will be less than the demand

rate, and the queue of customer orders will grow without bound. The following simple example illustrates how this may occur.

A simple two-stage production line (Figure 5.2), receives orders from customers at an average rate of 1 order every 60 minutes. The line is balanced, and there is no travel time for product between stations. Since each station requires 45 minutes of processing time to produce one part, the bottleneck rate (rate of the slowest station), r_b, of this system is 1/45 units/min. Since the system can't produce any faster than the slowest station, the bottleneck rate is the maximum possible throughput for this line.

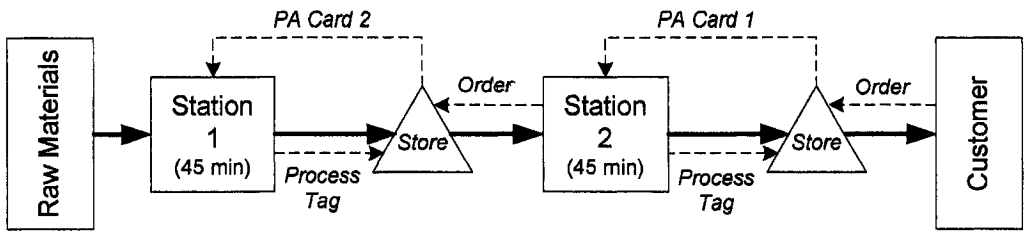

Figure 5.2: Two-stage Production Line Example

According to Hopp and Spearman (2001), the critical amount of work in process inventory required for such a system to achieve a maximum theoretical throughput equal to the bottleneck rate, r_b, is given by

$$W_0 = r_b T_0$$

where

W_0 = critical WIP level, and
T_0 = raw processing time (sum of the mean processing time at all stations).

When the amount of WIP in the system is less than the critical WIP level, W_0, then the maximum theoretical throughput of the system is given by

$$TH = \frac{w}{T_0}$$

where

TH = maximum theoretical throughput, and
w = amount of WIP in the system.

For this system, the raw processing time, T_0, is 90 minutes, resulting in a critical WIP level, W_0, of 2 units. Therefore, if $w = 1$, the maximum possible throughput of this line is 1 unit every 90 minutes. The single unit of WIP would have to be processed at the first station while the second sat idle, and then processing would take place at the second station while the first station sat idle. This throughput rate would be insufficient to meet the arriving customer demand.

The WIP in this system would be limited to one if, using the PAC system, the number of process tags at Station 2 were set to one. When a customer order arrived, a PA card for the final product is created if the process tag is available. The process tag is tied up with this PA card until the authorized product has been completed. If there is no initial inventory at the supplying store, then this process tag is tied up until Station 1 produces and delivers required component, and Station 2 produces the final product. No new jobs can enter the system during this time. Thus, the throughput of this line will be limited to 1 unit every 90 minutes, which is less than the arrival rate of customer orders.

Therefore, it would appear that the minimum amount of process tags at the second station must be at least two. However, Hopp and Spearman's equations provide the theoretical maximum throughput rates; if processing times or demand arrivals are variable, of if material requires travel time between cells, then two process tags may still be insufficient. For complex systems with assembly cells and several products, the determination of these minimum WIP values (and thus minimum number of process tags) is difficult to determine analytically.

Therefore, experimentation with the simulation model is necessary to determine a sufficient number of process tags (particularly at downstream stations) to ensure feasibility. Any combination of PAC parameters with less than these values would be considered infeasible, and the system would never reach steady state. Customer wait times would increase with time, and the interdeparture time of finished product would be less than the interarrival time of demands.

5.2.2 Choice of Maximum Parameter Values

Just as the minimum values for parameters must be addressed, the maximum values to use are also an issue. Using extreme parameter values will have a negative impact on one or more of the system measures without a complementary improvement in any other measure. As a simple example, when customer fill rate is close to 100% at a finished goods store, increasing the initial inventory level (z) at that store will result in a higher average finished goods inventory without improving the fill rate by any significant amount. If extreme points were excluded from the training data, we would not be able to say with any confidence that the neural network metamodel could estimate performance at these points; however, given the effect on performance, it is unlikely that these points would be of any interest in the analysis of the system.

5.2.3 Rules for Selection of Combinations of PAC Parameters

In the following section, we address the issue of the selection of combinations of PAC parameters. Buzacott and Shanthikumar (1993) discuss some of the requirements for these parameters; we have articulated these in the first three rules outlined below. The final two rules were inspired by a discussion in Bielunska-Perlikowski (1997), with regard to the selection of parameters in systems where batching is permitted. She observed that it is possible to select parameter combinations which will cause a halt in production, due to the limitations on process tags and the requirements for the formation of batches for production. She presented two rules to avoid this problem; however, we have determined that these rules did not cover all possible scenarios where this problem may occur. Therefore, we have developed the final two rules to ensure that, in systems where batching is permitted, the combination of process tag, batch, and initial inventory parameters are chosen so as to ensure a feasible system.

5.2.3.1 Number of Upstream Process Tags

Referring to Figure 5.3, Buzacott and Shantikumar (1993) state that the maximum possible number of active PA cards at an upstream cell, Cell U, is equal to the number of process tags at the requesting downstream cell, Cell D, plus the initial inventory at Store

U. Therefore, the maximum number of process tags allocated to Store U need not exceed this amount. If additional process tags were available, they would never be used.

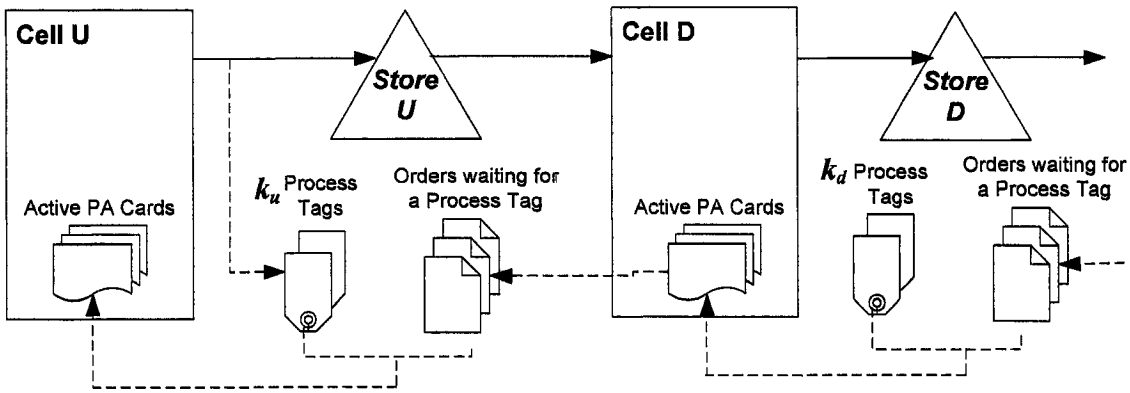

Figure 5.3: Flow of Process Tags, Orders and PA Cards

The result is that without affecting the performance of the system, we may restrict the number of process tags at an upstream cell

Rule 1: (Buzacott and Shanthikumar, 1993). *The number of process tags at an upstream cell/store (for each product) should not exceed the maximum possible number of outstanding orders for the downstream part.*

If a PA card at Cell D triggers an order for a single part from Store D, then the rule is expressed as:

$$k_u \leq k_d + z_u \qquad [48]$$

If a PA card at Cell D triggers n orders for the part at Store U, then this limit must be adjusted so that the right hand side is equal to the maximum number of outstanding orders for the downstream part:

$$k_u \leq nk_d + z_u \qquad [49]$$

The reason that this is relevant is that any design point which violates this rule is not necessary in the training set. Including these points unnecessarily increases the size of

the training set, thereby increasing the complexity of the network training. Therefore, it may be possible to eliminate several level combinations from the training set to reduce training complexity and time. Although the networks could be trained without imposing this rule, the resulting points are not of any interest; the performance at these points is equal to the performance at the point where the number of process tags at the upstream station is equal to the maximum possible number of outstanding orders at the cell.

5.2.3.2 Cells with Multiple Machines

When a cell has more than one processing machine, then there must be sufficient process tags available at a cell/store so that all machines may be utilized. Therefore this leads to Rule 2:

Rule 2: (Buzacott and Shanthikumar, 1993). *The sum of process tags at all stores associated with a production cell must be equal to or greater than the number of machines at that cell.*

For each store, this rule is expressed as follows:

$$\sum_{I} k_i \geq c_j \qquad [50]$$

where

c_j = number of processing machines (servers) at cell j
I = set of products produced at cell j

5.2.3.3 Batching Rules

In order for a batch of PA cards to be formed at a cell, r orders must arrive and each order must be paired with a process tag. This leads to the following rule:

Rule 3: (Buzacott and Shanthikumar, 1993). *The value of the batch parameter, r, at any cell must be less than or equal to the number of process tags at the cell.*

This rule is expressed as follows:

$$k_d \geq r_d \qquad [51]$$

where

r_d = the batch size at the downstream cell.

The next two rules developed were inspired by some observations of Bielunska-Perlikowski (1997). Since a store cannot form a PA card unless a process tag is available, nor can that process tag be released from the PA card until the authorized product has been completed, then the process tags, in a sense, could be regarded as a resource. These resources are allocated to orders in order to form PA cards. At the downstream cell, if no process tags are available (or not enough to form a batch of PA cards), then no PA cards can be formed and no orders can be sent to the upstream cell until the required process tags return to the store. If, at the same time, processing at the upstream cell is halted because the cell is waiting for more orders from the downstream to form a batch of PA cards, then we will encounter a "deadlocking" situation. Wysk et al. (1991) discuss this phenomenon in FMS systems, and define deadlocking as the state where "parts are assigned to various machines in a manufacturing system such that any further part flow is inhibited." (Wysk et al., 1991, p. 853). In this situation, the flow of information (in the form of orders) becomes inhibited, and ultimately results in a halt to the flow of parts.

In order to prevent the stoppage of production, the batch values have to be set such that it will eventually be possible to form a batch of PA cards at both cells at all times. Assuming a store supplies only one downstream cell, there are two scenarios which must be addressed. The first is when the batch size for PA cards at an upstream station, r_u, is less than the batch size at the downstream station, r_d. When a batch of PA cards is issued from Store D to Cell D, it will trigger the release of r_d orders to Store U. Since $r_u < r_d$, one or more batches of PA cards may be formed and sent to Cell U, possibly leaving some orders waiting at Store U. In order to ensure another batch of PA cards will be formed at Cell U, there must be sufficient product produced at the cell and sent to Store U so that, when processing takes place at Cell D, a sufficient number of process tags will be released, and another batch of PA cards will eventually be formed at Store D. This will trigger additional orders to be sent to Store U, thus enabling additional batches of PA cards to be formed. Therefore, in order to ensure production will not be halted, the following inequality must hold:

$$(k_d - r_d) + z_u + \left\lfloor \frac{r_d}{r_u} \right\rfloor r_u \geq r_d \quad if \; r_d > r_u \qquad [52]$$

The left hand side of the inequality represents the maximum number of process tags that will eventually become available at Store D after the issuance of a batch of PA cards to Cell D. The first term, $(k_d - r_d)$, is the number of process tags remaining at Store D after a batch of PA cards has been issued to Cell D. If there is any initial stock in Store U (z_u), then these components will be delivered to Cell D and processed, thus eventually releasing the same number of process tags to the store. The final term represents the number of PA cards that will be created and sent to Cell U upon receipt of the orders generated by the r_d PA cards issued to Cell D; these will eventually be processed at Cell U, sent to Store U, and finally sent to Cell D, where they will be processed, thus releasing process tags upon receipt of the finished parts at Store D. This number of process tags must equal or exceed the batch size at Store D, in order for subsequent batches of PA cards to be formed at Store D.

If the issuance of a PA card to Cell D triggers more than one order, n, for the same component part from Store U, then equation [52] must be adjusted. Cell D requires n parts from Store U in order to produce one product and release one process tag. When r_d PA cards are created, Cell D generates nr_d orders for the component part at Store U. Upon receipt of these orders, $\lfloor nr_d / r_u \rfloor r_u$ PA cards will be issued to Cell U, and eventually these parts will arrive at Store U. Combined with the initial inventory at that store, z_u, there will be one process tag released to Store D for every n components available for processing through Cell D. Therefore, the inequality may now be written:

$$(k_d - r_d) + \left\lfloor \frac{1}{n}\left(z_u + \left\lfloor \frac{nr_d}{r_u} \right\rfloor r_u \right) \right\rfloor \geq r_d \quad if \ r_d > r_u \qquad [53]$$

If the batch size at Cell U exceeds that of the Cell D ($r_u > r_d$), then a different problem may cause a halt in production. The system must be able to create a sufficient number of orders for Store U to create a batch of PA cards for Cell U. Since orders are generated by Cell D upon receipt of PA cards, then Cell D must receive a sufficient number of PA cards in order to generate the necessary orders. When the first batch of PA

cards is formed and sent to Cell D, the orders that will be generated and sent to Store U will not be sufficient to trigger the generation of PA cards for Cell U. Depending on the batch sizes, it will take two or more batches of PA cards to be delivered to Cell D before a sufficient number of orders are sent to Store U. Therefore, PA cards that cannot be completed at Cell D due to a lack of parts in storage at Store U must wait until an additional batch of PA cards is created downstream, thus generating more orders for materials from Store U. During this time, process tags from Store D are tied up in the PA cards awaiting processing at Cell D. If there are not enough process tags available at Store D to produce the additional batch(es) of PA cards (once sufficient orders have been received), production will be halted.

When $r_d = 1$, the maximum number of orders which may be outstanding at Store U is equal to $k_d + z_u$. However, when $r_d > 1$, the maximum number of outstanding orders at Store U becomes a multiple of r_d. Therefore, the constraint which must be satisfied in this case is:

$$\left\lfloor \frac{k_d + z_u}{r_d} \right\rfloor r_d \geq r_u \quad \text{if } r_d < r_u \qquad [54]$$

If Cell D requires n components from Store U to produce one product, then each time a PA card is created at Cell D, n orders are sent to Store U. If there is initial inventory at Store U, then this will result in the production of one product at Cell D (and thus the release of one process tag) for every n components in the store. Therefore, equation [54] becomes:

$$\left\lfloor \frac{k_d + \lfloor z_u / n \rfloor}{r_d} \right\rfloor n r_d \geq r_u \quad \text{if } r_d < r_u \qquad [55]$$

If a cell/store which does not batch orders supplies a cell/store where batching is permitted, then $r_d > 1$, and $r_u = 1$. In this case, the first inequality, equation [53], applies, and since $r_u = 1$, it reduces to $k_d + \lfloor z_u / n \rfloor \geq r_d$. However, Rule 3 already requires that $k_d \geq r_d$; therefore this rule need not be applied in this case. If the supplying cell/store does batch orders received from a downstream cell/store where such batching is

permitted, then $r_d = 1$, and $r_u > 1$. The second inequality, equation [55], applies and reduces to $nk_d + n\lfloor z_u/n \rfloor \geq r_u$. Rule 1 already requires $k_u \leq nk_d + z_u$, and Rule 2 requires $k_u \geq r_u$; therefore, combining these two inequalities leads to

$$nk_d + z_u \geq k_u \geq r_u$$
$$\therefore nk_d + z_u \geq r_u \qquad [56]$$

If $n = 1$, then equations [55] and [56] reduce to the same inequality. However, if $n > 1$, then equation [56] will be binding, and must be applied. Finally, if $r_u = r_d$, then it can be shown that both inequalities are the same. Therefore, the overall result is that this rule on batch sizes must only be considered for each cell/store which is supplied by a cell/store where batching is permitted. When a cell produces more than one product, this rule must be applied to each of the products.

Rule 4: *For each cell/store combination which is downstream from (supplied by) a cell/store where batching is permitted, the following inequalities must hold for the case where the upstream store supplies the product to only that downstream cell:*

$$\left\lfloor \frac{k_d + \lfloor z_u/n \rfloor}{r_d} \right\rfloor nr_d \geq r_u \quad \text{if } r_d < r_u \qquad [55]$$

$$(k_d - r_d) + \left\lfloor \frac{1}{n}\left(z_u + \left\lfloor \frac{nr_d}{r_u} \right\rfloor r_u \right) \right\rfloor \geq r_d \quad \text{if } 1 < r_u \leq r_d \qquad [53]$$

where

n = the number of orders for product generated by the downstream cell upon receipt of a single PA card (which is equal to the quantity of the upstream product required for the production of one unit at the downstream cell).

Finally, when a cell/store where batching is permitted supplies the same product to more than one downstream cell, orders for the product will be received from more than one source. Therefore, the maximum number of orders which can be transmitted to the upstream cell (before the upstream cell replenishes the store through production) is equal to the sum of the maximum number of orders each downstream cell can create. If there is

initial inventory at the upstream cell, then it will be sent downstream upon receipt of requisitions, on a FIFO basis. There is no way to know in advance which downstream cell will be transmitting the first batch.

If we assume for a moment that there is no initial inventory in the upstream store, then the system is only feasible if the following is true:

$$r_u \leq \sum_p n_p \left\lfloor \frac{k_p}{r_p} \right\rfloor r_p \qquad [57]$$

where

r_p = batch size at the p^{th} station downstream from Cell/Store U
k_p = the number of process tags at the p^{th} station downstream from Cell/Store U
n_p = the number of products cell p demands from the upstream cell each time a PA card is created at cell p

If the inequality in equation [57] is true, then the system is feasible with the current parameters. If it is not true, then the system may still be feasible, provided $z_u > 0$. Referring to the discussion for Rule 1 (Section 5.2.3.1), the maximum number of orders a downstream cell can generate depends upon z_u and k_d; the initial inventory can supply some (or all) of the initial orders, thus eventually resulting in the release of process tags at the downstream station (once production with the required parts has been completed at the downstream cell), and potentially enabling more batches of orders to be created. The problem in this case is that the initial inventory will be sent to one (or possibly more than one) of the downstream stores, but we cannot know in advance which one will receive this inventory. Therefore, in order to determine if the system is feasible, we must ensure that no matter how the initial orders arrive, every possible scenario is feasible.

The first step in this process is to determine whether or not the initial inventory at the upstream store will eventually allow for the downstream cell to form additional batches of PA cards, and thus for additional orders to flow upstream. It is possible that the entire initial inventory will be released to any of the downstream cells – it depends on which cell transmits the first batch. Therefore, we must determine whether or not every

downstream cell can form additional batches of PA cards if it receives the initial inventory from upstream. The following calculation, similar to the one developed for Rule 4, will enable this determination:

$$A = \min_{p} \left\{ n_p \left\lfloor \frac{k_p + \lfloor z_u / n_p \rfloor}{r_p} \right\rfloor r_p - n_p \left\lfloor \frac{k_p}{r_p} \right\rfloor r_p \right\} \qquad [58]$$

The first term represents the maximum number of orders which will be sent upstream given that cell p receives the initial inventory, z_u. The second term is the maximum number of orders, assuming that the cell does not receive this inventory. If $A = 0$ for any cell/store p, then we know that the receipt of the initial inventory will not result in any additional batches of PA cards to be formed, and therefore no more orders at the upstream store. Therefore, since it is possible that cell p may receive the initial inventory, and because we have already determined that the inequality in equation [57] is not true, then we know that a halt in production may occur given these parameters. Therefore, we would consider this combination of parameters infeasible.

If $A > 0$, then each downstream store can create at least one new batch of orders if it receives some or all of the z_u units. Therefore, we must determine the minimum number of additional orders that can be created as a result. If $z_u \leq n_p r_p \ \forall p$, then no matter which downstream cell/store sends the first batch of orders, the inventory in the upstream store will be immediately depleted. Therefore, A represents the minimum number of additional orders which could be created. Whether this will be sufficient to form a batch at the upstream cell must now be checked by adding the A orders to the right hand side of the original inequality (equation [57]):

$$r_u \leq \sum_{p} n_p \left\lfloor \frac{k_p}{r_p} \right\rfloor r_p + A \qquad [59]$$

If equation [59] holds, then the system is feasible. If it does not hold, and $z_u \leq n_p r_p \ \forall p$, then the system is infeasible. However, if the inequality in equation [59]

does not hold, but $z_u > n_p r_p$ for at least one p, then the determination of the minimum number of additional orders possible becomes more difficult. Assume that the q^{th} cell/store creates the first batch of PA cards, r_q, and therefore transmits $n_q r_q$ orders upstream. If $z_u > n_q r_q$, then there will still be inventory left in the upstream store after all requested product is sent to cell/store q. This remaining product will be sent to the next cell/store requesting product, and will influence whether or not this next cell/store will be able to form additional batches. Thus, in order to find the minimum number of additional orders possible, one would have to evaluate all possible sequences of order batch arrivals at the upstream store. Instead, we will present a rule which will simply guarantee that the selection of parameters will result in a feasible system.

Rule 5: *When a cell/store where batching is permitted supplies more than one downstream cell with the same product, and the downstream cells also permit batching, the following inequalities must hold in order to ensure that the system is feasible:*

$$r_u \le \sum_p n_p \left\lfloor \frac{k_p}{r_p} \right\rfloor r_p + \min_p \{A(p)\} \qquad [60]$$

where

$$A(p) = \begin{cases} 0 & \text{if } z_u = 0 \\ n_p \left\lfloor \dfrac{k_p + \lfloor z_u / n_p \rfloor}{r_p} \right\rfloor r_p - n_p \left\lfloor \dfrac{k_p}{r_p} \right\rfloor r_p & \text{if } 0 < z_u < n_p r_p \\ n_p r_p & \text{if } z_u \ge n_p r_p \end{cases} \qquad [61]$$

In the special case where batching is not permitted at the downstream stations ($r_p = 1$ for all p), this rule must still be applied to ensure production will not be halted at the supplying cell.

5.3 Experimental Design

Even by limiting the range of possible input values as discussed above, there can still be a very large number of combinations of these input values. Producing a very large

training data set means longer simulation time, and longer training time. However, in order to build reasonably accurate metamodels, the training set must adequately cover the state space. Therefore, a space-filling experimental design method (Kleijnen et al., 2005) should be used to select design points for the training dataset. Because we expect a nonlinear, monotonic relationship between the input factors and output measures, it would seem logical to use a 3-factorial experimental design for the production of the dataset. In this situation, a low, mid-range, and high value for each input point would be chosen, and the training set would include every combination of each of these co-ordinates. This would result in 3^k design points, where k is the number of co-ordinates within a design point. However, Hurrion and Birgil (1999) presented an alternate approach for choosing the design points. They first identified the minimum and maximum values, and therefore the range of acceptable values, for each co-ordinate in the design point. To construct a design point, a value for each co-ordinate was randomly chosen from the acceptable range. They then trained neural networks with 2, 3 and 4-factorial data sets, as well as with randomly designed datasets of the same size. Their results showed that the networks trained with the randomly chosen datasets were more accurate and efficient than those trained with full-factorial designs. Alam et al. (2004) found that a Latin Hypercube design supplemented with domain knowledge outperformed both the full-factorial design and the random sampling design.

We chose to use a form of stratified sampling (Helton and Davis, 2003) as the sampling technique. In this approach, the range of acceptable values for each co-ordinate is sub-divided into three smaller subsets, covering the entire range. Normally, the ranges would be chosen such that each range is roughly the same size. Each subset is then designated a coordinate level, and a factorial design is then used to generate the level combinations for each design point. If there are k coordinates (input variables) for each design point, then there could be 3^k design points generated; however, if for any level combination there is no way to choose a design point which would satisfy the rules outlined in Section 5.2.3, then this combination is eliminated. Once the valid level combinations for design points have been determined, then a value for each coordinate is

randomly chosen from within the designated level, ensuring that each randomly chosen point does not violate the rules. To ensure complete coverage of the design space, two more points are then added to this dataset – one with every co-ordinate set at its minimum feasible value, and one with every co-ordinate set at its maximum value in the high range.

Of course, as the number of input variables increased, the number of design points to be simulated increases. Even though PACSIM is relatively fast, this approach may prove too time consuming as the number of input variables increases. In this situation, we would recommend either reducing the number of levels to two per variable, or exploring a form of Latin Hypercube design.

Another problem may be encountered when the number of input variables is small (less than 5). As discussed in Section 4.5, the size of the training set should well exceed the number of network weights. Assuming each network will have only one output node, the formula for the number of weights in a neural network with a single hidden layer is given by:

$$W = J(I+2)+1 \qquad [62]$$

where

I = the number of input variables, and
J = the number of hidden nodes.

Therefore, depending on the number of input variables, the number of training point generated according to the sampling procedure discussed above may not exceed the number of training. In order to avoid this problem, more than one design point may be randomly chosen at each level combination, or the number of levels may be increased, in order to increase the size of the training dataset. In the examples presented in Chapter 6, the training datasets contained at least twice as many points as there were network weights.

5.4 Generating Performance Estimates using PACSIM

With a set of design points chosen, the next step is to simulate each design point and obtain steady-state estimates of performance using PACSIM. The run parameters which

must be set are the length of each simulation replication, the number of replications per design point, and the length of the warm-up period.

5.4.1 Selecting Run Length and Number of Replications

The accuracy of the simulation responses increases as the length of the simulation and/or the number of simulation replications per design point is increased. This accuracy is dependent upon the variance of the output of the simulation. However, given the space-filling experimental design we have chosen, we expect that even with "noisy" estimates, the networks will be able to fit the data to a reasonable degree of accuracy. The simplest approach therefore would be to run only one replication per design point.

5.4.2 Transient Analysis

The system performance measurements used to train the networks must be observations of the system operating at steady state; because the PACSIM model starts with an empty system, early measurements of performance will not necessarily be representative of the system operating at steady state, and therefore must be removed. The challenge is in determining the point in time when the system has reached steady state. Welch's graphical procedure (see Law and Kelton, 2000) is one technique commonly used to determine this point.

The average steady state performance measure is defined as the average observation after l initial observations have been removed:

$$\bar{Y}(m,l) = \frac{\sum_{i=l+1}^{m} Y_i}{m-l} \qquad [63]$$

Welch's technique involves running n simulation replications (usually at least 5 – 10) and computing the average of the observations from each replication:

$$\bar{Y}_i = \frac{1}{m}\sum_{i=1}^{m} Y_{ji} \qquad [64]$$

where

Y_{ji} = the i^{th} observation from the j^{th} simulation replication, $i = 1,\ldots,m$ and $j = 1,\ldots,n$.

A moving average of these observations is then computed and plotted. The moving average window, w, is chosen such that w is a positive integer and $w \leq \lfloor m/4 \rfloor$. The moving average is then calculated as:

$$\bar{Y}_i(w) = \begin{cases} \dfrac{\sum_{s=-w}^{w} \bar{Y}_{i+s}}{2w+1} & \text{if } i = w+1,\ldots,m-w \\ \dfrac{\sum_{s=-(i-1)}^{i-1} \bar{Y}_{i+s}}{2i-1} & \text{if } i = 1,\ldots,w \end{cases} \quad [65]$$

These averages, $\bar{Y}_i(w)$, are plotted against the values of i. The point at which the plot converges is considered the point at which the simulation has reached steady state, and the number of initial observations to delete is then determined from this graph.

In order to determine a suitable warm-up period for the PACSIM model, the minimum feasible design point should be simulated; the reason for this is that it is assumed that this system will have the longest queues, and therefore will take the longest (of all combinations of PAC parameters) to reach steady state. Therefore, any decision on a warm-up period for this PAC parameter combination would be adequate for all PAC combinations in the training set. The performance measure used for the analysis should be the one which is expected to take the longest to reach steady state; in these problems, the customer service delay, which will be based on the length of the customer order queue, has been used for this analysis. Once the number of observations is determined using the above technique, the point in time is approximated by multiplying this number of observations, l, by the average time between demand arrivals.

5.5 Designing and Training Neural Network Metamodels

This framework involves the use of feedforward neural networks with a single hidden layer and the logistic function as the transfer function. As previously mentioned in Section 4.3.2, the typical approach when dealing with multiple responses is to construct a separate neural network metamodel for each performance measure of interest.

Since we have chosen this approach, each network to be constructed has only one output node, and all networks for a particular model will be trained with the same training dataset. Networks are trained using the Levenberg-Marquardt algorithm (Section 4.4.3).

The approach selected for determining the number of hidden nodes for each network is a simple constructive approach (Kwok and Yeung, 1997). This approach involves training a network with only a few hidden nodes (relative to the number of inputs), and then training additional networks with more nodes until an acceptable level of error is achieved. The following error measures were calculated for each trained network:

Mean Squared Error (*MSE*):
$$MSE = \frac{1}{N}\sum_{n=1}^{N}(v_n - t_n)^2 \quad [66]$$

Mean Error(*ME*):
$$ME = \frac{1}{N}\sum_{n=1}^{N}(v_n - t_n) \quad [67]$$

Mean Absolute Error (*MAE*):
$$MAE = \frac{1}{N}\sum_{n=1}^{N}|v_n - t_n| \quad [68]$$

where

N = number of design points in the training dataset,
v_n = the neural network output for the n^{th} design point, and
t_n = the observed (simulated) output for the n^{th} design point.

As well, the *postreg* function in MATLAB produces a plot of the performance observations from the training set along the x-axis, and the corresponding network output along the y-axis (called post regression plots). It also fits a linear regression line to these points, and computes the equation for this line, and the regression coefficient, R. If the neural network output mapped exactly to the training set, then this line would have a slope of one and a y-intercept of zero. If either the regression coefficient, R, or the slope of the regression line is far from 1, or if some of the points lie far from this line, then it is possible that the network did not have enough hidden nodes to learn the mapping. A plot was generated for each network.

The process of adding nodes and retraining was continued until an acceptable level of error was reached and the post regression plots were considered acceptable.

5.6 Metamodel Analysis

The trained neural networks are deterministic function approximations of the system performance measures given the PAC control parameter settings. Therefore, there are many possibilities available to analyze this system performance. Some of these are outlined below.

5.6.1 Optimization of a Cost Function

If the goal of the analysis was to minimize the overall cost of a control strategy, then a deterministic cost function of the system performance, using the neural network function approximations for those performance measures, may be used. Costs such as holding costs for inventory, or penalties for late orders, would be the coefficients of this cost function. The advantage to using the neural network metamodels for cost function optimization is that, if the decision maker wanted to change any of the cost parameters, the process could easily be repeated, but without having to retrain any of the networks.

In situations where the number of input combinations are relatively small ($<10^6$ points), one approach to determining a global optimal solution is to evaluate all possible input combinations using the neural networks and determine the optimal result. Where the state space is quite large, a heuristic such as simulated annealing or tabu search may be directly applied, using the networks to compute the function evaluation at each point in the search.

A nonlinear programming technique designed for continuous variables may be applied, but the result will most likely be non-integer. Exploring the integer neighbours of this optimal solution may result in a relatively good solution, but there is no guarantee that the optimal integer solution resides in this neighbourhood. However, such a technique may be applied if only to determine a good starting point for a heuristic technique.

The analysis may be an iterative process. For example, it is possible that the true optimal solution to the minimization of a cost function lies in an area where the cost function is somewhat flat, and therefore optimization using the metamodels may produce a result which is close to, but not necessarily, optimal. The neighbourhood of this solution would be a 'region of interest' (Figure 5.4). More simulation experiments would be carried out in this region, and new network metamodels would be trained (but these new models would only be valid for the neighbourhood from which the training points were drawn). The result will be even more precise metamodels, and then optimization of a function of these new metamodel functions may well lead to a more accurate result.

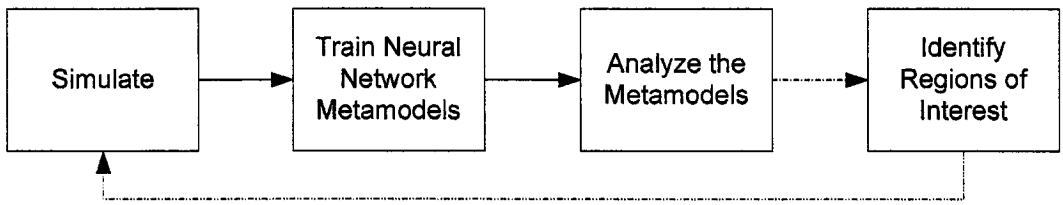

Figure 5.4 Flowchart of Possible Analysis using Neural Network Metamodel

5.6.2 Optimization of One Performance Measure Subject to Constraints

Another possibility is to specify constraints for all but one of the output measures, and then minimize (or maximize) the remaining output measure subject to those constraints. For example, if a minimum fill rate and maximum cycle time were specified, then the metamodel can be used to find the set of input parameters which minimize the average finished goods inventory. Once again, every eligible design point can be evaluated using the networks, or a heuristic search algorithm may be applied to find the optimal solution.

5.6.3 Understanding the Effects of PAC Parameters

The neural network metamodels may also be used to gain a better understanding of the impacts of the control parameters on the performance of the system. Scenario analysis (evaluation of one point) can be quickly carried out with the neural network.

However, it may be more interesting to learn more about the effects of the points in general on the performance of the system. For example, how does increasing the initial inventory parameter at the final store change the average finished goods inventory? One approach to this is to calculate the partial gradients of the neural network function with respect to the input variables at the point in question. The difference in magnitude of these gradients should provide a better understanding of which parameters have the largest impact on each performance measure at a particular point.

5.6.4 Constructing Exchange Curves

As discussed in Section 3.5.3, exchange curves are very useful tools for deciding on the appropriate policy when the costs for performance measures, such as penalties for late orders or inventory carrying charges, are not known. To construct useful exchange curves, two conflicting measures (or composite measures, such as total inventory) must be selected. Depending on the complexity of the manufacturing system (and therefore the size of the networks), it may be feasible to evaluate the performance measures for all possible combinations of PAC parameters and select the non-dominated points. When this is not feasible, a heuristic such as simulated annealing applied to a parameterized objective function should be applied to find the points on the exchange curves, using the approach discussed in Section 3.5.3.

5.6.5 Evaluation of a Traditional Strategy

With the neural network metamodels, one can also evaluate the performance of a traditional control strategy, such as Kanban, by constraining the combinations of design parameters (according to Buzacott and Shantikumar, 1992), and evaluating only these combinations using the network metamodels. For example, to represent a Kanban strategy, all initial inventory parameters (z_i) must equal the number of process tags (k_i) at each cell/store combination. No batching is permitted, and there is no requisition delay. The results of the neural network evaluations can then be used to determine the optimal strategies, whether through the use of a cost function, optimization of one performance measure subject to constraints on the other measures, or a combination of both.

5.7 Concluding Remarks

Our framework provides the ability to analyze and optimize the selection of PAC control scheme parameters in several different ways. In the next chapter, we will demonstrate the application of this framework to some example manufacturing models.

CHAPTER 6

EXAMPLE MODELS AND RESULTS

In this chapter we demonstrate the application of this framework and several examples of the analysis discussed in Chapter 5. Throughout this chapter, five different manufacturing systems are used in various experiments.

Section 6.1 provides more details of the experimental environment and the implementation of the framework for a two-stage serial production line, Model 0. Details of the construction of the training datasets and the neural network metamodels are provided for this system. Similar procedures were followed for all example systems discussed in this chapter.

Section 6.2 discusses the results of optimizing a cost function of the performance measures for this model using OptQuest compared to using the neural networks. A three-stage serial production line with material travel times, Model A, is introduced in Section 6.3, and we show how the neural network metamodels trained for this system may be used to optimize a single performance measure (in this case, minimize finished goods inventory), subject to constraints on the other variables. In Section 6.4, a two cell production system, Model B, is introduced. This system has one cell that produces two parts and a second cell that assembles those components. This model is used to show how the networks can be used to estimate performance, even when setup and travel time is required – even for this small system, analytical models do not exist.

Section 6.5 discusses Model C, a three-cell system with one cell capable of producing two parts supplying two downstream cells, each producing a final product. The networks trained for this system are used to study the impact of batching on system performance. Section 6.6 shows how a simulated annealing algorithm can optimize a cost function of the system performance.

Section 6.7 demonstrates how exchange curves can be constructed using trained neural networks. In the first example, exchange curves are constructed for Model A, but only for parameter combinations fitting the PAC definition of a Kanban policy. This is

then compared to an exchange curve constructed without this limitation. In both cases, all possible policies were evaluated using the neural networks. For the more complex Model C, simulated annealing was used to construct an exchange curve for this system using the networks.

In Section 6.8, we address the issues of accuracy and suitability of neural networks for this framework by conducting several experiments with an even more complex manufacturing system, Model D. The generalization ability of the networks is demonstrated in Section 6.8.1 by showing that the error measurements for a set of data not used in the training set is only slightly higher than the results seen from the training set itself. We also demonstrate that the networks can be trained to produce unbiased estimates of the expected value functions in an experiment discussed in 6.8.2. In Section 6.8.3, we construct second and third order polynomial regression models from the same training data, and show that the MSE of the trained networks outperforms both the second and third order models in almost every case.

6.1 Experimental Environment

This section provides an overview of the experimental procedures used to apply the framework to the example manufacturing systems. The PACSIM model was written in FORTRAN77, and Absoft ProFortran 7.5 was used to compile and execute the programs on a 2.0 GHz AMD2400 computer. The training datasets produced by the simulation model were imported into MATLAB Version 7.0 (R14), where the Neural Network Toolbox routines were used to train the networks. The experiments involved several stages, which are described in the sections below (for more details, see Appendix A).

We illustrate these stages using Model 0 (Figure 6.1), which has two production cells in series and produces a single product. Characteristics of this model are:

- Demand for the final product arrives according to a Poisson distribution with a mean time between arrivals of 60 minutes.
- Processing time distribution at each station is Weibull with α=2 and mean $t_p(i)$.
- Raw materials and requisitioned parts are immediately available (no travel time).

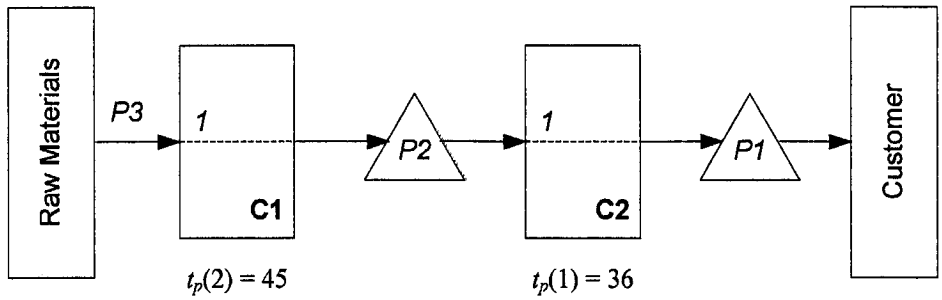

Figure 6.1: Model 0

Table 6.1: Model 0 System Parameters

Product	Production Cell	Inputs (Part and Qty)	Mean Processing Time (min)
1	C2	P2 (Qty 1)	36
2	C1	P3 (Qty 1)	45

6.1.1 Initializing the PAC Simulation Model

The first stage of each experiment involved providing the PAC general performance simulation model with the required information about the system to be simulated, such as number of cells/stores and processing times. This information is provided to the simulation model in the text file, *pact.in*. In order to automate this process, a spreadsheet application was built to allow for entry of this information in a form, and then, through the use of a macro, the parameter file for the simulation was generated and saved as a text file (see Appendix A). It was assumed for these models that customer orders are never cancelled; therefore, no cancellation or correction tags were required. None of the example models have by-product production, and therefore surplus tags were not required. This process is common to all models discussed in this chapter.

6.1.2 Determining Feasible Design Points (Preprocessing)

The next step in the process involves the determination of the minimum values of the control parameters required to ensure the manufacturing system could achieve the required throughput, as discussed in Section 5.2.1. The process involved creating a test file of design points with the initial inventory parameters all set to zero, and the number of process tags at each station set to very low (usually 1) values. These design points

were simulated several times each using PACSIM. A new program, *preprocess.f*, was created to assist in determining whether the simulation was able to reach steady state for these parameter combinations. This program reads in the output files from the simulation runs (including the information provided by the new subroutine SSTATE) and produces a report with the following information:

- the slope of the linear regression line of the independent variable (system time) and the dependent variable (customer delay time).
- the average customer delay time for the entire simulation run, along with the average of delays in the first half and the second half of the simulation run.
- the average time between product arrivals at the finished goods store(s) and the average time between customer order arrivals.

For Model 0, the design point $z_1 = 0$, $k_1 = 1$, $z_2 = 0$, and $k_2 = 1$ was simulated, and then the preprocessing report generated for this point (Figure 6.2).

Preprocessing Report
Created on: 16-Mar-05

Number of RUNS per Design Point: 10
Length of each simulation run: 250 days

Design point tested 0 1 0 1

Product No. 1

Run	Time BTW Departs	Time BTW Orders	Percent Diff	Reg Line	Customer Delay Time Average	1st Half	2nd Half	Cycle Time
1	81.549	59.405	-37.28	1.37787	66648.2	42131.4	91343.3	81.6
2	80.708	59.640	-35.32	1.32922	64185.0	40562.7	88180.9	80.7
3	81.326	59.060	-37.70	1.34214	64323.7	40146.7	88337.2	81.3
4	80.916	60.292	-34.21	1.26905	62186.9	40890.2	83977.7	80.9
5	81.150	61.813	-31.28	1.23617	60075.8	38725.9	81425.7	81.2
6	81.326	61.405	-32.44	1.34819	65944.1	43211.6	88676.5	81.3
7	81.134	59.653	-36.01	1.31548	63684.9	40346.7	87044.1	81.1
8	81.343	61.503	-32.26	1.25127	60888.0	39476.7	82145.0	81.3
9	80.727	59.198	-36.37	1.37942	66143.1	42382.5	89544.0	80.7
10	81.419	59.200	-37.53	1.34182	64541.2	41177.4	87558.7	81.4
	Average:		-35.04	1.31906	63862.1	40905.2	86823.3	81.2

Percent Difference less than -0.200%
Avg. Regression slope above 0.150
2nd half delay average is 112.3% higher 1st half delay average

***Design point probably INFEASIBLE ***

Figure 6.2: Excerpt from the Preprocessing Report (Model 0)

In this example, the simulation of the selected design point resulted in a time between departures much higher than the time between arrivals for product; the customer delay time appeared to be increasing and the slope of the linear regression line was quite high; these all indicated that the system was unable to achieve the required throughput, and therefore this point would be considered infeasible. For Model 0, a subsequent test with the process tags for product 1 set to two did achieve the throughput rate, and therefore established the minimum parameter values. For more complex models, several experiments were sometimes necessary to determine this point. More details on this program may be found in Appendix A. The minimum values for the process tag parameters was determined through this process to be $k_1 = 2$, $k_2 = 1$.

6.1.3 Transient Analysis

Prior to generating the training data for the model, transient analysis was conducted using the customer delay time as the measure for analysis. The minimum values for process tags established through the preprocessing experiments were used for the design point, as discussed in Section 5.4.2. Using Welch's graphical technique (see Figure E.1, Appendix D), it was determined that a 50 day warm up period for Model 0 was sufficient to ensure the simulation had reached steady state.

6.1.4 Verification of the Simulation Model

Prior to generating the training data for Model 0, we used the model to verify the PACSIM model. This procedure was only performed for this model. Using exponential distributions for the processing times at both cells, queuing formulas were used to find the expected performance measures. The simulation model was run with all values of k set to very large numbers, so that all information about customer orders would immediately be passed up to the first cell. Initial inventory at both cells were set to zero. Therefore, as soon as a customer order arrives at this system, a job would be released at Cell 1.

Because orders arrived at Cell 1 according to a Poisson process, and processing time at Cell 1 was exponential, then departures from Cell 1 (and therefore arrivals at Cell 2)

would also be exponentially distributed (e.g. Ross, 1997). Therefore, both cells can be modeled as M/M/1 queues, and basic queuing formulas can be used to predict the average total time in the system (Table 6.2).

Table 6.2 Queuing formulas and calculations for Model 0

Measure	Formula	Cell 1	Cell 2	Total
Processing Rate (parts/minute)	μ	1/45	1/36	
Arrival Rate (parts/minute)	λ	1/60	1/60	
Utilization	$\rho = \frac{\lambda}{\mu}$	0.75	0.6	
Average number of parts in cell	$L = \frac{\rho}{1-\rho}$	3	1.5	4.5
Average waiting time in queue and in process (minutes)	$W = \frac{L}{\lambda}$	180	90	**270**
Average waiting time in the queue (minutes)	$W_q = W - \frac{1}{\mu}$	135	54	189
Average number of parts in the queue	$L_q = \lambda W_q$	2.25	0.9	3.15

Total time in the system therefore should be 270 minutes. Forty replications of the simulation were executed. Using the replication-deletion method for means (Law and Kelton, 2000), the average cycle time, *CT*, was determined to be 269.86 minutes, and the 95% confidence interval for the true mean of the average cycle time was [265.48, 274.24] (see Appendix D). Since this includes 270 minutes, the simulation model was assumed to be correct.

6.1.5 Generating the Training Dataset

With the minimum feasible values for the design points established, the next stage involved generating the design points to be included in the training dataset.

Because Model 0 is small, choosing only a single point from each level combination would result in, at most, only $3^4 = 81$ design points in the training set, which may be fewer than the number of weights in the model. Therefore, three points were selected from each level combination.

The only applicable process tag rule was Rule 1, resulting in the following constraint on the number of process tags at cell/store 2:

$$k_2 \leq k_1 + z_2 \qquad [69]$$

A program, *generate_random.f* (Appendix A for description, and Appendix I for code), was created to generate these points, based on the ranges for each coordinate and the process tag rules for the system (Section 5.2.3). The program requires the number of parameters and the ranges for each, and the number of desired replication for each level combination. For each input parameter, three levels for the values were chosen (low, mid-range, and high), such that the ranges covered the entire range of values for the given parameter (Table 6.3). The ranges were designed to be roughly the same size.

Table 6.3: Model 0 Design Point Ranges

	Low	Mid	High
z_1	0,2	3,6	7,10
k_1	2,4	5,7	8,10
z_2	0,2	3,6	7,10
k_2	1,3	4,6	7,10

Therefore, there was the potential to generate 3^k design points, where k is the number of input parameters, which would represent all possible combinations of levels for the input parameters. However, for some of these level combinations, there would be no way to randomly choose any point which would satisfy the process tag rules (as discussed in Section 5.2.3). Therefore, these level combinations were dropped. The program would then randomly generate a point for each level combination. If the point followed the rules, it was added to the dataset; if not, another point would be randomly chosen and this process repeated until a valid point was found.

The resulting dataset for Model 0 included 236 design points – 234 generated randomly and the upper and lower bounds for the input space. These design points were then simulated using PACSIM to generate performance estimates for the system.

6.1.6 Training the Neural Network Metamodels

Following the procedure discussed in Section 5.5, the neural networks for these experiments were trained using the MATLAB Neural Network Toolbox. A separate network was trained for each of the performance measures of interest. The Levenberg-Marquardt algorithm (*trainlm*) was used for training. The input and output data were normalized to values between -1 and +1 prior to training, and then converted back to actual values afterwards. The logic of this algorithm is described in Hagan et al. (1996) and Appendix C.

A separate neural network was trained for each of average customer delay, average cycle time, customer fill rate, and average finished goods inventory for Product 1 (Figure 6.3). The trained network parameters are shown in Table 6.4.

Table 6.4: Network Parameters, Model 0

Network	Hidden Nodes	Training Epochs
Customer Delay	12	300
Cycle Time	15	200
Fill Rate	15	250
FGI	12	150

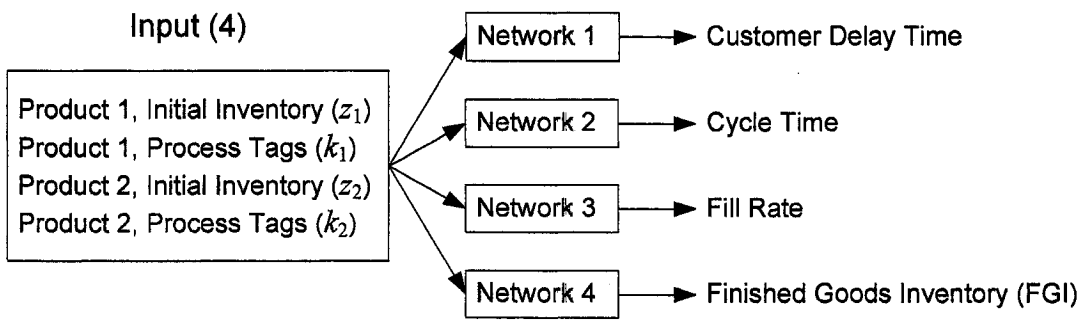

Figure 6.3: Neural Networks for Model 0

A plot of the performance measures from the dataset versus the network performance measures was generated using the *postreg* function in MATLAB (as discussed in Section 5.5). The x-axis (T) of this plot is the value from the training set, and the y-axis (A) is the

network response. The equation for the linear regression line fit to these points is shown at the top of the graph, and the regression coefficient, *R*, is shown in the upper left corner. The plot for the Cycle Time network (Figure 6.4) shows that the network fits the data very well. The plots for the other networks for this model are in Appendix D. These post regression plots were similarly constructed for all networks in this chapter.

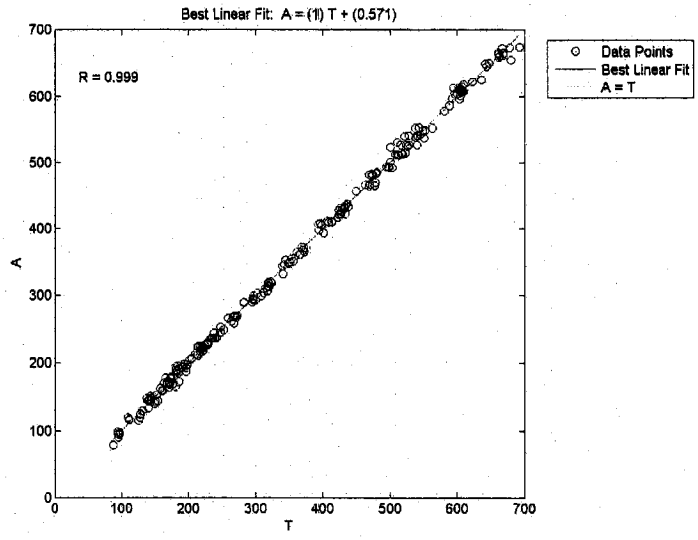

Figure 6.4: Post Regression Plot for Cycle Time Network, Model 0

The error calculations for each network are shown in Table 6.5, along with the minimum and maximum observations for each measurement in the training data set.

Table 6.5: Training Results, Model 0

Network	Minimum Observation	Maximum Observation	MSE	Mean Error	Mean Absolute Error
Customer Delay	0	325.07	49.718	-0.0476	4.5459
Cycle Time	87.64	693.09	44.716	0.4936	5.1399
Fill Rate	0	1.000	0.00048	5.4E-05	0.0161
FGI	0	8.917	0.01360	1.6E-05	0.0927

Three design points not in the training set were chosen and presented as input to the network. Design point 1 represents a Kanban system with three Kanbans at each

cell/store. Design point 2 represents a CONWIP system with the maximum number parts in the system set at six. Design point 3 is also a Kanban system, but with two Kanbans at the first cell and four at the second cell. These design points were also simulated 10 times, and a 99% confidence interval for the expected was computed (Table 6.6).

Table 6.6: Some Neural Network and Simulation Results for Model 0

			Simulations		
			LCL	UCL	NN*
Design Point 1	$z_1 = 3$, $k_1 = 3$, $z_2 = 3$, $k_2 = 3$	Delay	13.8	25.7	18.3
		Cycle Time	233.3	238.8	229.6
		Fill Rate	0.756	0.814	0.764
		FGI	1.69	1.81	1.76
			Simulations		
			LCL	UCL	NN*
Design Point 2	$z_1 = 6$, $k_1 = 6$, $z_2 = 0$, $k_2 = 6$	Delay	12.4	26.1	16.7
		Cycle Time	161.7	169.9	166.6
		Fill Rate	0.823	0.870	0.870
		FGI	3.13	3.34	3.34
			Simulations		
			LCL	UCL	NN*
Design Point 3	$z_1 = 4$, $k_1 = 4$, $z_2 = 2$, $k_2 = 2$	Delay	11.3	19.9	14.3
		Cycle Time	171.5	174.9	173.2
		Fill Rate	0.810	0.865	0.812
		FGI	2.36	2.53	2.33

Since we do not have the true expected values for each of these functions, we must estimate them by constructing confidence intervals. Obviously, tighter confidence intervals could be constructed by collecting more samples. In Table 6.6, all but two of the results fall within the confidence intervals; in these cases, we cannot reject the hypothesis that the neural network result equals the expected value. For Design Point 1, at a 99% confidence level, the true expected value of the Cycle Time lies between 233.3 and 238.8 minutes; the network estimated 229.6 minutes. It can then be said in this case, at a 99% confidence level, that the relative error between the true expected value and the network estimate is between -1.58% and -3.85%.

6.2 Optimization using Neural Networks and Simulation Optimization

Model 0 was used to demonstrate the optimization of a cost function. The following function, C, is a cost function with respect to the performance measurements:

$$C = C_d\left(\frac{v_d}{60}\right) + C_c\left[\left(\frac{v_c}{60}\right) - 1.35\right] + C_r D(1 - v_r) + C_i v_i$$

where

C = System Cost ($/hour)
C_d = Cost of customer delay ($/hour)
v_d = Average customer delay time (minutes)
C_c = Cost for cycle time beyond processing time ($/hour)
v_c = Average product cycle time (minutes)
C_r = Cost for orders not filled immediately upon arrival ($/order)
D = demand rate (orders/hour)
v_r = Customer fill rate (percentage of orders filled upon arrival of demand)
C_i = Cost for inventory ($/unit/hour)
v_i = Average finished goods inventory (units)

The value of 1.35 in the second term is the total (in hours) of the processing time required on the machines; thus, any value of v_c above 1.35 represents waiting time in the system. The values for the cost variables used in this example were C_d = $10/hour, C_c = $1/hour, C_r = $20/order, C_i = $1/unit/hour, and D = 1 unit/hour.

The goal was to find the PAC parameters which minimized this cost function with respect to lower and upper bounds on the parameters, but no other constraints. In order to determine the optimal solution to this problem, a dataset consisting of all of the possible combination of PAC parameters was generated. The size of this dataset was further reduced by limiting the number of process tags at the first cell to one, as any other value served only to increase cycle time without any improvement in customer service, as there was no travel time from raw materials to the first cell. All points with a cost greater than $10 (as estimated by the neural networks) were eliminated, resulting in a dataset containing 327 points; each of these points was then simulated 20 times. In practice, this would rarely be a feasible approach, but since the model is very small this took just under

17 minutes on a 2.0 GHz 2400AMD computer. The best results from this experiment are reported in Table 6.7, where they are ranked in order of average cost.

Table 6.7: Top Results from Simulations at Each Point

Point				Simulation (20 Runs)		
z_1	k_1	z_2	k_2	LCI	Average	HCI
6	8	2	1	$ 6.85	$ 7.41	$ 7.97
7	10	1	1	$ 6.93	$ 7.44	$ 7.94
8	9	0	1	$ 7.00	$ 7.50	$ 8.00
6	9	2	1	$ 7.23	$ 7.51	$ 7.79
8	10	0	1	$ 7.10	$ 7.53	$ 7.97
8	4	0	1	$ 7.15	$ 7.54	$ 7.94
7	4	1	1	$ 7.10	$ 7.55	$ 8.00
8	10	1	1	$ 7.36	$ 7.61	$ 7.86
9	7	0	1	$ 7.21	$ 7.61	$ 8.02
7	4	2	1	$ 7.39	$ 7.62	$ 7.84

Note that even after 20 simulations, the 99% confidence intervals for the true mean are quite wide; in fact, none of these points is statistically different. Of the 327 points simulated, 120 points have a 99% confidence interval for the expected value which covers part of the interval for the first point in this table.

All of the points were also evaluated using the neural networks, and the cost was calculated from the results. The top results are shown in Table 6.8, along with the corresponding results for 20 replications at each point.

Table 6.8: Top Results from Neural Network Evaluations

Point				Simulation (20 Runs)			
z_1	k_1	z_2	k_2	NN Result	LCI	Average	HCI
6	2	2	1	$ 7.30	$ 7.41	$ 7.98	$ 8.55
6	2	3	1	$ 7.32	$ 7.25	$ 7.87	$ 8.49
6	3	2	1	$ 7.35	$ 7.13	$ 7.62	$ 8.12
5	2	3	1	$ 7.40	$ 7.25	$ 7.85	$ 8.45
6	3	3	1	$ 7.42	$ 7.50	$ 7.76	$ 8.01
6	4	2	1	$ 7.45	$ 7.59	$ 8.16	$ 8.73
5	2	4	1	$ 7.46	$ 8.18	$ 8.63	$ 9.08
5	3	3	1	$ 7.46	$ 7.47	$ 8.09	$ 8.72
7	2	2	1	$ 7.50	$ 7.39	$ 7.75	$ 8.11
7	2	1	1	$ 7.51	$ 7.90	$ 8.34	$ 8.79

A simulation model of this system was also built in Arena 9.0 (Appendix D). The model was verified in the same manner as the FORTRAN simulation, using exponential processing times, zero initial inventories, and unlimited process tags. The OptQuest function was then used to find the optimal parameter settings subject to the same cost function. The optimization procedure was permitted three simulations at each tested point, and each simulation ran for 250 days (less the 50 day warm up period). The top 10 solutions from this procedure, along with the corresponding simulation results, are reported in Table 6.9.

Table 6.9: Top 10 Results from OptQuest Optimization Procedure, Model 0

Point				Optquest	Simulation (20 Runs)		
z_1	k_1	z_2	k_2	Cost	LCI	Average	HCI
7	3	1	1	$ 6.88	$ 7.11	$ 7.74	$ 8.38
6	3	1	1	$ 7.07	$ 7.07	$ 7.70	$ 8.33
7	4	1	1	$ 7.12	$ 7.10	$ 7.55	$ 8.00
7	5	1	1	$ 7.15	$ 7.12	$ 7.72	$ 8.32
7	9	1	1	$ 7.15	$ 7.42	$ 8.17	$ 8.91
7	8	1	1	$ 7.15	$ 7.19	$ 7.69	$ 8.18
7	10	1	1	$ 7.15	$ 6.93	$ 7.44	$ 7.94
7	7	1	1	$ 7.15	$ 7.08	$ 7.75	$ 8.43
8	3	1	1	$ 7.22	$ 7.32	$ 7.84	$ 8.36
7	6	1	1	$ 7.32	$ 7.35	$ 7.93	$ 8.51

Optquest reported an optimal result at $6.88, which is much lower than the optimal result reported by the networks. This point was simulated 20 times using the original Arena model, which resulted in an average cost of $7.99, and a 99% confidence interval of (7.47, 8.52). Figure 6.5 shows the results of 20 replications at this point using PACSIM as well as Arena. The neural network estimation of the mean for this point was $7.53. The reason that OptQuest reported such a low value may be explained by the spread of the simulation results. Given the relatively high cost associated with customer delay, a difference of only a few minutes has a significant effect on the total cost. At each point during the procedure, OptQuest performs three replications, and the average of these three is taken as the value for this point. Given the fact that there are so many points that cannot be declared to be significantly different even after 20 replications, and the fact that OptQuest was only permitted three replications at each point, it is logical to assume that the likelihood of OptQuest seeing three consecutive "low" results for one of these many points would be very high.

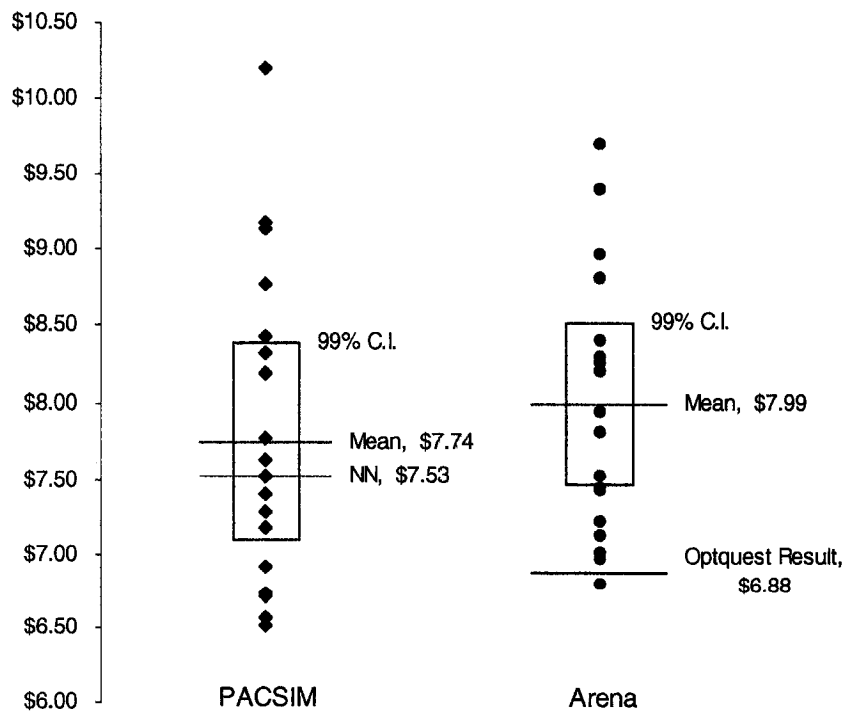

Figure 6.5: Comparison of Simulation Replications at OptQuest Optimal Point

This experiment illustrates the problem in using simulation optimization, as discussed in Section 3.5.1. By treating the simulation observation (or even the average of a few simulations) at a point as a true expected value, there is no way to guarantee true optimality, without conducting a very large number of simulation experiments at every point.

6.3 Minimization of a Single Performance Measure Subject to Constraints

This example involves the determination of the PAC parameters required to minimize the average finished goods inventory of Model A (Figure 6.6), subject to constraints on the other performance measures. This system has three processing stations in series. Demand for the final product arrives according to a Poisson distribution with a mean time between arrivals of 60 minutes. The processing times have a Weibull distribution (with $\alpha = 2$). The mean processing times at each station are shown in Figure 6.6. While raw materials are instantly available at cell 1, parts 2 and 3 require 5 minutes of travel time (fixed) to arrive at the downstream station after they have been requisitioned.

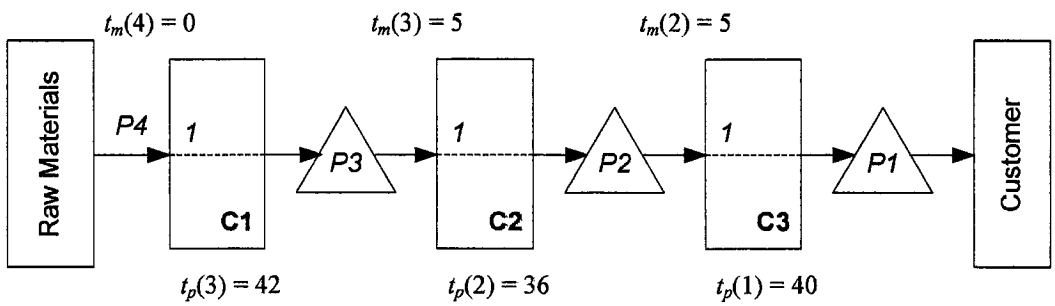

Figure 6.6: Model A

Minimum and maximum values were chosen (as in previous models), and the ranges for each parameter were then selected (Appendix E). The training dataset for this example model, using three levels per variable, would have generated 729 points. However, the following constraints were applied to the dataset:

$$k_2 \le k_1 + z_2$$

$$k_3 \le k_2 + z_3$$

This resulted in 677 design points in the training set. Four neural networks (Figure 6.7) were trained using this data (see Appendix E for the post regression plots). A MATLAB *m*-file (Appendix I) was used to count the number of valid combinations of parameters in the input space. For this model, it was determined that the networks would be valid for 1,043,196 parameter combinations.

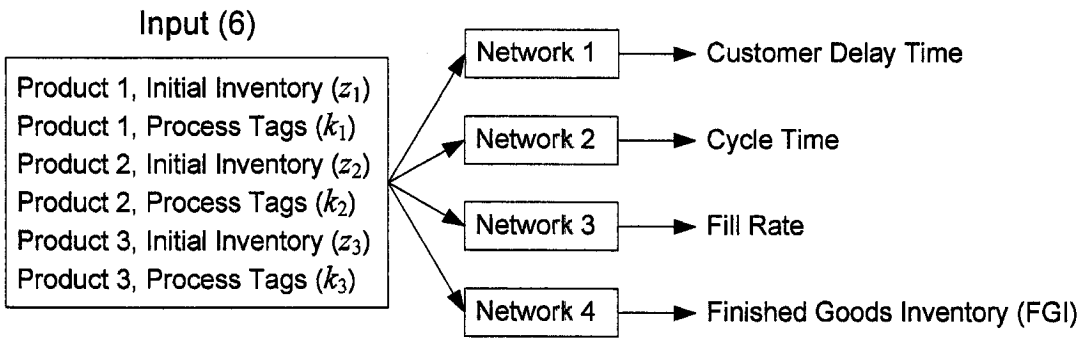

Figure 6.7: Neural Networks for Model A

The networks were then used to find the PAC parameters which minimized the average finished goods inventory level for the system, subject to constraints on the other measurements:

- Average customer delay time less than or equal to 60 minutes,
- Cycle time less than or equal to 300 minutes, and
- Fill rate equal to or greater than 90%.

The optimization problem is formulated as follows:

$$\min\ z = v_4$$

$$s.t.$$

$$v_1 \le 60$$

$$v_2 \le 300$$

$$v_3 \ge 0.90$$

$$k_2 \le k_1 + z_2$$

$$k_3 \le k_2 + z_3$$

where

v_1 = output of network 1 - customer delay time (minutes)
v_2 = output of network 2 - cycle time (minutes)
v_3 = output of network 3 - fill rate
v_4 = output of network 4 - finished goods inventory (units)

The purpose of the last two constraints on this problem is to constrain the search to the space covered by the networks, as discussed in Section 5.2.3.

The approach to finding the minimum point for this simple model was to simply evaluate all eligible points using the neural networks. The top 20 results from these network evaluations were chosen and each simulated 20 times (see Appendix E). The cycle time constraint was satisfied (on average) for all of these points, but the fill rate constraint was not in all cases. Therefore, for each point, a 99% confidence interval for the mean fill rate and finished goods inventory level was constructed. Table 6.10 contains the points where the confidence interval for the mean fill rate contains 0.90, and therefore it cannot be concluded that the constraint is violated. In all cases, the finished goods estimate is within the confidence interval, and in all but four cases, the fill rate value is within this interval. Of these four cases, the maximum relative error occurs at the second point in the table, where that the network result differs (at a 99% confidence level) from the true expected value by, at most, 2.92%.

Table 6.10: Best Results from Neural Network Evaluation of All Points, Model A

Design Points						Network		Simulation			
								Fill Rate		FG Inv	
z_1	k_1	z_2	k_2	z_3	k_3	Fill Rate	FG Inv	Avg.	99% C.I.	Avg.	99% C.I.
5	10	2	3	2	1	0.904	3.268	0.899	[0.888, 0.911]	3.264	[3.216, 3.313]
5	6	2	8	2	1	0.912	3.261	0.897	[0.888, 0.907]	3.268	[3.222, 3.315]
5	4	2	5	2	1	0.916	3.280	0.902	[0.890, 0.915]	3.269	[3.218, 3.320]
5	8	2	8	2	1	0.904	3.260	0.904	[0.895, 0.913]	3.276	[3.237, 3.315]
5	4	2	6	2	1	0.912	3.269	0.898	[0.888, 0.908]	3.281	[3.238, 3.324]
5	5	2	7	2	1	0.910	3.269	0.901	[0.890, 0.912]	3.284	[3.243, 3.324]
5	10	2	2	2	1	0.909	3.275	0.902	[0.891, 0.914]	3.286	[3.233, 3.339]
5	10	2	8	2	1	0.904	3.260	0.900	[0.889, 0.911]	3.289	[3.249, 3.328]
5	8	2	6	2	1	0.907	3.272	0.902	[0.890, 0.914]	3.292	[3.238, 3.347]
5	7	2	6	2	1	0.912	3.280	0.904	[0.895, 0.912]	3.294	[3.242, 3.347]
5	7	2	7	2	1	0.909	3.269	0.906	[0.898, 0.915]	3.305	[3.271, 3.339]
5	9	2	8	2	1	0.901	3.259	0.908	[0.898, 0.917]	3.310	[3.264, 3.357]
5	9	2	5	2	1	0.904	3.270	0.907	[0.900, 0.915]	3.311	[3.275, 3.347]
5	6	2	7	2	1	0.911	3.272	0.910	[0.899, 0.921]	3.312	[3.261, 3.362]
5	7	2	8	2	1	0.910	3.261	0.910	[0.900, 0.920]	3.314	[3.270, 3.358]
5	8	2	7	2	1	0.903	3.265	0.907	[0.896, 0.918]	3.314	[3.264, 3.364]
5	9	2	4	2	1	0.909	3.278	0.906	[0.897, 0.915]	3.320	[3.284, 3.356]
5	8	2	5	2	1	0.911	3.282	0.910	[0.899, 0.921]	3.320	[3.271, 3.369]
5	5	2	6	2	1	0.913	3.281	0.911	[0.902, 0.921]	3.329	[3.294, 3.364]
5	10	3	8	1	1	0.900	3.264	0.916	[0.910, 0.923]	3.365	[3.338, 3.393]

Since many of these confidence intervals overlap, the optimal result is still unclear. However, upon examination of the results in Table 6.10, there are similarities in the best results. The only values which differ amongst the top results are the number of process tags at Cells 1 and 2. The top solutions have values for k_1 between 4 and 10, and k_2 has values between 2 and 8. This would indicate that the number of process tags at these two cells do not have much impact on the performance of the system, given the values of the other variables. It is straightforward to derive the partial derivatives of the network functions with respect to the input values (Appendix B). The evaluations of those derivatives at the top point are shown in Table 6.11.

Table 6.11: Partial Derivatives of the Network Functions at a Single Point

	Network		
	CT	Fill Rate	FGI
z_1	-0.5	0.049	0.950
k_1	-0.3	0.001	-0.001
z_2	39.2	0.040	0.194
k_2	-0.1	0.007	0.012
z_3	50.3	0.023	0.106
k_3	30.9	0.004	0.002

The difference in magnitude of these derivatives indicates the effects of changing the parameters from the current values. Note, as expected, that increasing the initial inventory parameter at the finished goods store will increase the average finished goods inventory, but increasing the number of process tags will have virtually no impact on this outcome, nor on the fill rate.

6.4 Comparing the Impact of Setup Time and Travel Time

For a manufacturing system with cells capable of producing more than one product, setup time at the cell would affect the performance of the system. To illustrate this, another model, Model B, was used. This model has only two processing stations, but one station produces two different products, which are both required by the second station for assembly (Figure 6.8). Customer demand for the assembled product, Product 1, arrives according to a Poisson distribution with mean time between arrivals of 60 minutes. Average processing times (with Weibull distributions, α =2) are also shown (Figure 6.8).

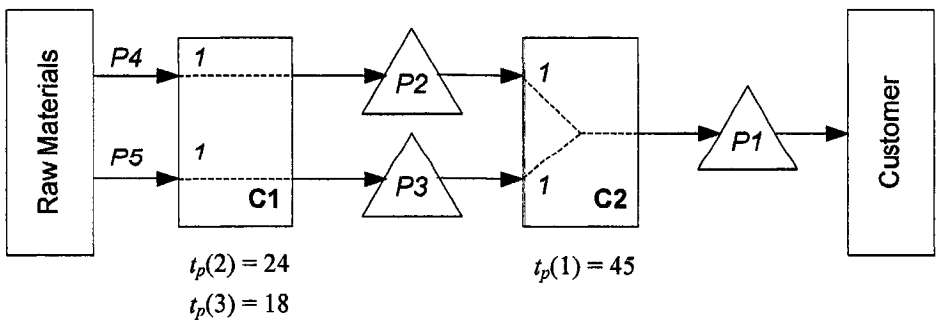

Figure 6.8: Model B

Because Cell 1 produces two different products, there may be setup time involved when switching between products. For this example model, two sets of networks were constructed; Case I, with no setup time required at Cell 1, and Case II, with setup time at Cell 1 and raw material travel time (which precludes any analytical modeling). Because there is an assembly operation in this model, cycle time was no longer a convenient measurement of performance, and therefore the work in process at each station for each product was used. Therefore, seven networks were constructed (Figure 6.9, Appendix F).

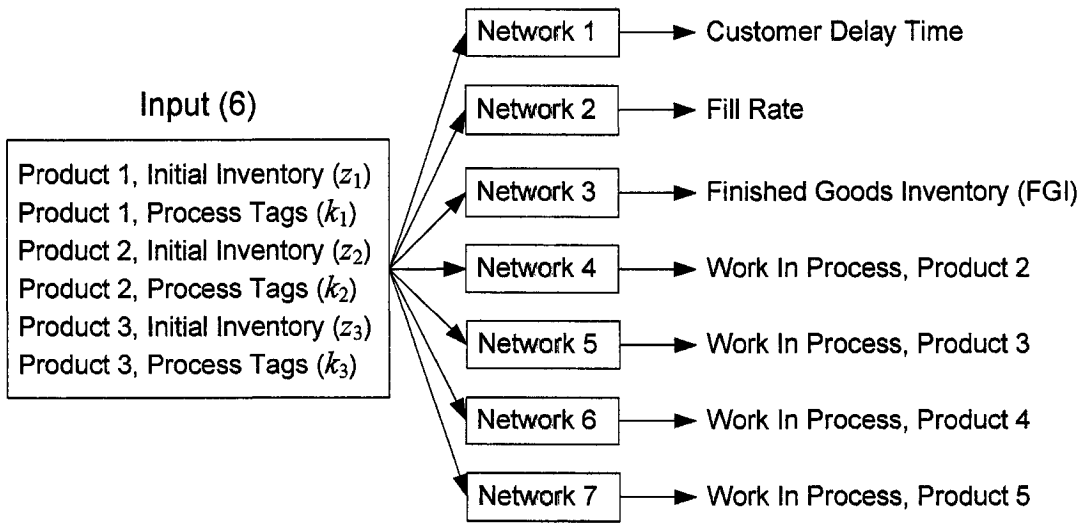

Figure 6.9: Neural Networks for Model B

6.4.1 Case I: No Setup or Travel Time

The training dataset was generated after selecting the minimum and maximum parameter values and after applying the following constraints on process tags (Rule 1):

$$k_2 \leq k_1 + z_2$$

$$k_3 \leq k_1 + z_3$$

With the lower and upper bounds included, there were 680 design points in the training dataset. Training results, including error measurements and the post-regression plots, are in Appendix F. Given the constraints and the ranges for the input values used

to generate the training set, a MATLAB program was used to determine that these networks are valid for 697,092 possible input parameter combinations.

Three design points were chosen to for network comparisons. The first represents a CONWIP system with $w = 8$. The second is a Kanban system with six Kanbans at the final assembly station and three for each of the subassemblies. The third example is a produce to order system with a limit of eight jobs in the system, and a limit of three jobs at the upstream station. The evaluations of these points using the neural network are shown in Table 6.12, as well as the results of 10 simulation runs at each point.

Table 6.12: Results for Sample Design Points, Model B, Case 1

Design Point 1		Simulations (10 Runs)		
		LCL	UCL	NN*
$z_1 = 8$	Delay	5.4	14.0	6.1
$k_1 = 8$	Fill Rate	0.900	0.934	0.950
$z_2 = 0$	FGI	4.72	4.95	4.80
$k_2 = 8$	WIP 2	1.10	1.26	1.6
$z_3 = 0$	WIP 3	0.81	0.96	1.12
$k_3 = 8$	WIP 4	0.79	0.88	0.81
	WIP 5	1.19	1.29	1.16

Design Point 2		Simulations (10 Runs)		
		LCL	UCL	NN*
$z_1 = 6$	Delay	6.5	20.5	15.1
$k_1 = 6$	Fill Rate	0.868	0.914	0.891
$z_2 = 3$	FGI	3.74	3.98	3.92
$k_2 = 3$	WIP 2	3.03	3.23	3.10
$z_3 = 3$	WIP 3	2.73	2.92	2.86
$k_3 = 3$	WIP 4	0.68	0.73	0.69
	WIP 5	1.02	1.08	0.99

Design Point 3		Simulations (10 Runs)		
		LCL	UCL	NN*
$z_1 = 0$	Delay	185.4	203.1	202.4
$k_1 = 8$	Fill Rate	0.000	0.000	0.008
$z_2 = 0$	FGI	0.00	0.00	0.04
$k_2 = 3$	WIP 2	1.07	1.26	1.18
$z_3 = 0$	WIP 3	0.77	0.96	1.13
$k_3 = 3$	WIP 4	0.65	0.69	0.67
	WIP 5	0.99	1.03	0.99

6.4.2 Case II: Setup and Travel Time

Model B, Case II, involved the same system parameters as Case I, but with a constant travel time for raw materials and a constant setup time at Cell 1 (Table 6.13). At Cell 1, the oldest PA card in the queue is processed first, regardless of the type of product being authorized. The same design points were used for this model, but simulated again with the new system parameters. Seven networks were trained, and results for the same example points are shown in Table 6.14, along with the average results from 10 simulation replications. As expected, these networks produce different results than those trained for Case I. For example, in Case I, for Design Point 1, the network reported an average fill rate of 95%; however, with additional travel and setup time at Cell 1 in Case II, the expected average fill rate dropped to 76.8% for the same control parameters.

Table 6.13: System Parameters for Model B, Case II

Product	Production Cell	Inputs (Part and Quantity)	Setup Time (Min)	Mean Processing Time (Min)	Move Time to Next Cell (Min)
1	2	2 (Qty 1), 3 (Qty 1)	--	45	--
2	1	4 (Qty 1)	5	24	--
3	1	5 (Qty 1)	5	18	--
4	--	--		--	5
5	--	--		--	10

Table 6.14: Results for Sample Design Points, Model B, Case II

Design Point 1		Simulations (10 Runs)		
		LCL	UCL	NN*
$z_1 = 8$	Delay	34.4	65.4	58.1
$k_1 = 8$	Fill Rate	0.715	0.809	0.768
$z_2 = 0$	FGI	3.28	3.76	3.06
$k_2 = 8$	WIP 2	0.79	0.91	1.28
$z_3 = 0$	WIP 3	0.41	0.52	1.14
$k_3 = 8$	WIP 4	2.13	2.50	2.20
	WIP 5	2.53	2.91	2.55

Design Point 2		Simulations (10 Runs)		
		LCL	UCL	NN*
$z_1 = 6$	Delay	21.3	68.4	32.6
$k_1 = 6$	Fill Rate	0.733	0.848	0.803
$z_2 = 3$	FGI	3.05	3.53	3.35
$k_2 = 3$	WIP 2	1.81	1.98	2.08
$z_3 = 3$	WIP 3	1.43	1.61	1.84
$k_3 = 3$	WIP 4	1.31	1.48	1.28
	WIP 5	1.52	1.66	1.61

Design Point 3		Simulations (10 Runs)		
		LCL	UCL	NN*
$z_1 = 0$	Delay	299.7	380.5	384.6
$k_1 = 8$	Fill Rate	0.000	0.000	0
$z_2 = 0$	FGI	0.00	0.00	0
$k_2 = 3$	WIP 2	0.82	0.90	1.41
$z_3 = 0$	WIP 3	0.44	0.51	0.86
$k_3 = 3$	WIP 4	1.35	1.52	1.13
	WIP 5	1.55	1.69	1.61

6.5 Analyzing the Impact of Batching

When a manufacturing system has a single cell capable of producing more than one product, there may be setup time required to switch between products (as discussed in Section 6.4). In order to avoid some of these setups, it may make sense to produce product in batches of more than one. This situation is explored through the use of another example model, Model C.

Model C has three processing cells, and two final products (Figure 6.10). Demand for the final product arrives according to a Poisson process with mean time between

arrivals of 60 minutes. Demand is then randomly assigned to Product 1 (40%) and Product 2 (60%). Processing times at the cells follow a Weibull distribution ($\alpha = 2$). Products 3 and 4 are both produced at Cell 1, where there is a 25 minute changeover time required, either to go from processing of product 3 to 4, or vice versa. Priority at Cell 1 for PA cards is first-in, first out. Raw material is available immediately upon requisition, but Products 3 and 4 require travel time to arrive at downstream stations. These move times, $t_m(i)$, and the mean processing times, $t_p(i)$, are shown in Figure 6.10.

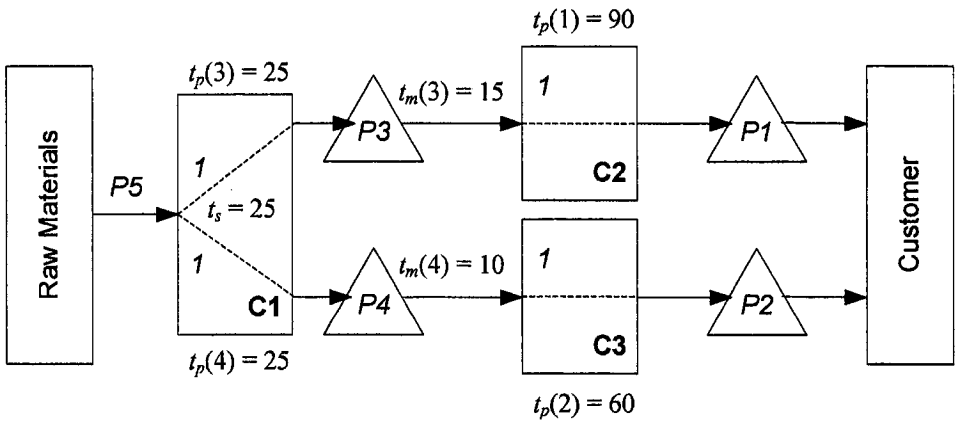

Figure 6.10: Model C

This model also allows for batch processing at Cell 1 ($r_3 \geq 1$, $r_4 \geq 1$). If the value of r for either product 3 or 4 is greater than one, PA cards authorizing production of that product are held at the store until a full batch is formed, and then the entire batch is sent to Cell 1 for processing. Because a batch of PA cards arrives at Cell 1 at the same time, once the first PA card in the batch is processed, all PA cards in the batch are processed in sequence, thus avoiding any changeover time during the batch run.

In order to form the training set for this model, valid combinations of initial inventory (z_i), process tags (k_i), and batch parameters (r_i) were generated as in previous models. The rules applied to choosing points for the training set were (from Rules 1, 3, and 4):

$$r_3 \leq k_3$$

$$r_4 \leq k_4$$

$$k_3 \leq k_1 + z_3$$

$$k_4 \leq k_2 + z_4$$

One point for each valid combination was generated, and then batch sizes for products 3 and 4 were randomly chosen for each point, such that the batch size for each product did not violate the rules above. Because both of the requesting cells had batch sizes of one, no further rules regarding batch values at cells 3 and 4 needed to be applied. In all, 6086 design points were chosen for the training set. These points were simulated for 200 days after the warm-up period of 50 days.

Because there are two final products and three other products produced or used by this system, nine neural networks were required to model each of the output measurements (Figure 6.11, Appendix G). Each network has 10 input values; the initial inventory and number of process tags for each cell/store, as well as the batch size for products 3 and 4 at Cell 1. Once again, given the constraints above and the ranges for the input values, it was determined that the networks would be valid for 828,214,960 possible combinations of input values.

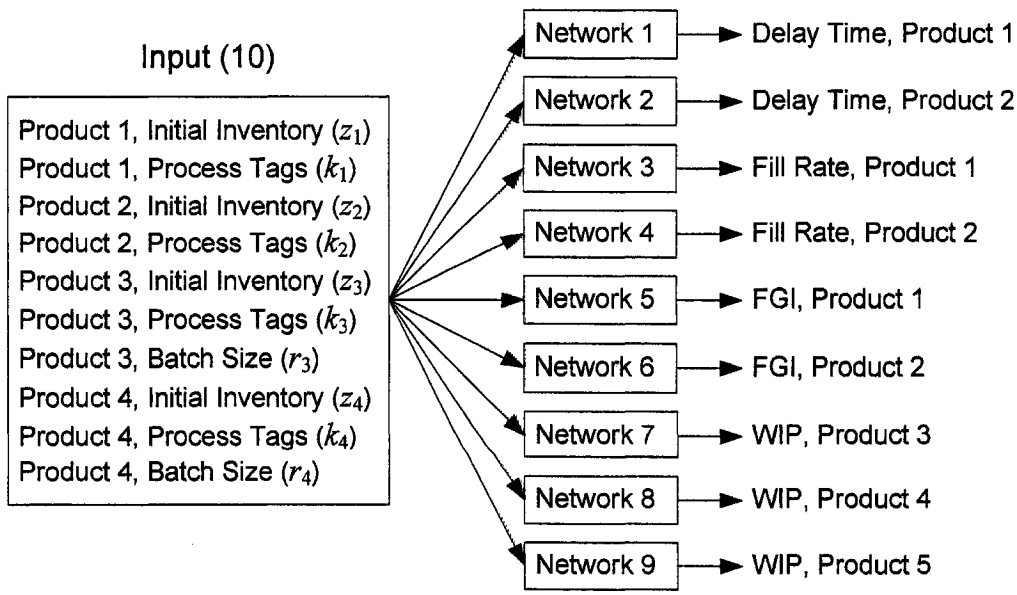

Figure 6.11: Neural Networks for Model C

The networks were then used to study the effects of batch processing at Cell 1 on the performance of the system. A configuration not found in the training dataset was chosen (Figure 6.12), and then the networks were used to evaluate the performance of the system for values of r_3 and r_4 between one and four. To show that the networks may be used for such analysis, each of these combinations was simulated 20 times. The complete results can be found in Appendix G.

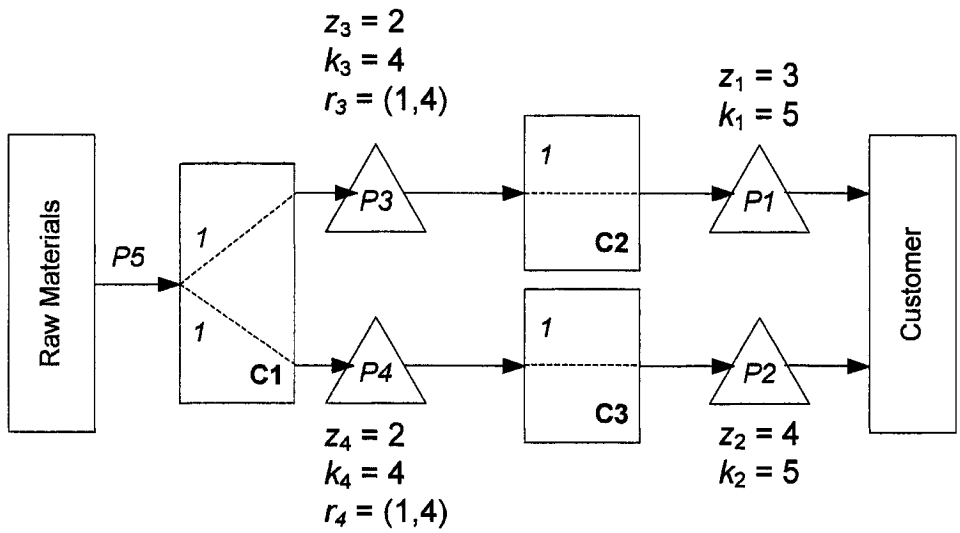

Figure 6.12: Configuration of Model C for Batch Size Analysis

Upon examination of the performance measures, it appeared that any change in the batch size for Product 3, r_3, did not have any appreciable effect on the performance measures for product 1, nor did changes to r_4 have any impact on Product 1. Examining these parameters separately, it can be seen from Table 6.15 that increasing the batch size for Product 3 at Cell 1 results in a decrease in customer fill rate for Product 1, but also a decrease in work-in-process inventory of Product 4. In particular, setting a batch size of two has only a small effect on fill rate and customer delay time, but a more noticeable impact on work-in-process inventory. These results, of course, are only valid for the configuration shown in Figure 6.12, but as can be seen from this example, the networks appear to be a fairly accurate approximation to the simulation results, and therefore could be used for such analysis.

Table 6.15: Performance with Different Batch Sizes, Model C, Product 2 (r_3 = 1)

	r = 1		r = 2		r = 3		r = 4	
	NN	Sim	NN	Sim	NN	Sim	NN	Sim
Cust. Delay	6.1	(6.0, 9.4)	7.1	(6.3, 9.3)	9.1	(8.4, 12.0)	14.8	(16.5, 19.3)
Fill Rate	0.929	(.907,.928)	0.921	0.904, 0.920	0.886	0.880, 0.900)	0.802	(0.807, 0.823)
FGI	2.80	(2.73, 2.80)	2.76	(2.68, 2.73)	2.52	(2.50, 2.55)	2.03	(2.02, 2.06)
WIP4	1.86	(1.84, 1.89)	1.35	(1.34, 1.38)	1.00	(0.97, 1.01)	0.90	(0.91, 0.94)
WIP5	0.59	(0.57,0.62)	0.62	(0.63, 0.66)	0.72	(0.71, 0.74)	0.84	(0.83, 0.86)

6.6 Optimization using Simulated Annealing

Model C has 828,214,960 eligible input combinations, therefore evaluating each possible combination using the networks is not a feasible approach for optimization. It is also worth noting that to simulate the 6086 design points in the training dataset took 13.010 minutes (see the PACSIM output report in Appendix G), for an average speed of 467 replications/minute. Even at this speed, it would take over 3 years to simulate all of these points, making simulation optimization virtually impossible. Therefore, as discussed in Section 5.6.1, a simulated annealing algorithm and the neural networks can be used to optimize a cost function of the performance measures of the system.

In this experiment, the goal was to find the design point which minimized the following cost function:

$$C = C_r D[(1-v_3)+(1-v_4)]+C_i(v_5+v_6)+C_{wip}(v_7+v_8+v_9) \qquad [70]$$

where

C = System Cost ($/day)
v_3 = output of network 3 (fill rate, product 1)
v_4 = output of network 4 (fill rate, product 2)
v_5 = output of network 5 (average finished goods inventory, product 1)
v_6 = output of network 6 (average finished goods inventory, product 2)
v_7 = output of network 7 (average WIP, product 3)
v_8 = output of network 8 (average WIP, product 4)
v_9 = output of network 9 (average WIP, product 5)
C_r = Cost for orders not filled immediately upon arrival ($/order)
D = average demand rate (orders/day)
C_i = Cost for finished goods inventory ($/unit/day)
C_{wip} = Cost for WIP inventory ($/unit/day)

A simulated annealing algorithm was written in MATLAB to optimize the function (see Appendix I). The logic of the algorithm is shown in Figure 6.13, and the parameters for the algorithm are shown in Table 6.16

Table 6.16: Parameters for the Simulated Annealing Algorithm

Number of Cooling Cycles	20
Initial Temperature	1000
Number of Iterations at First Temperature	5000
Number of Iterations at Subsequent Temperatures	2000

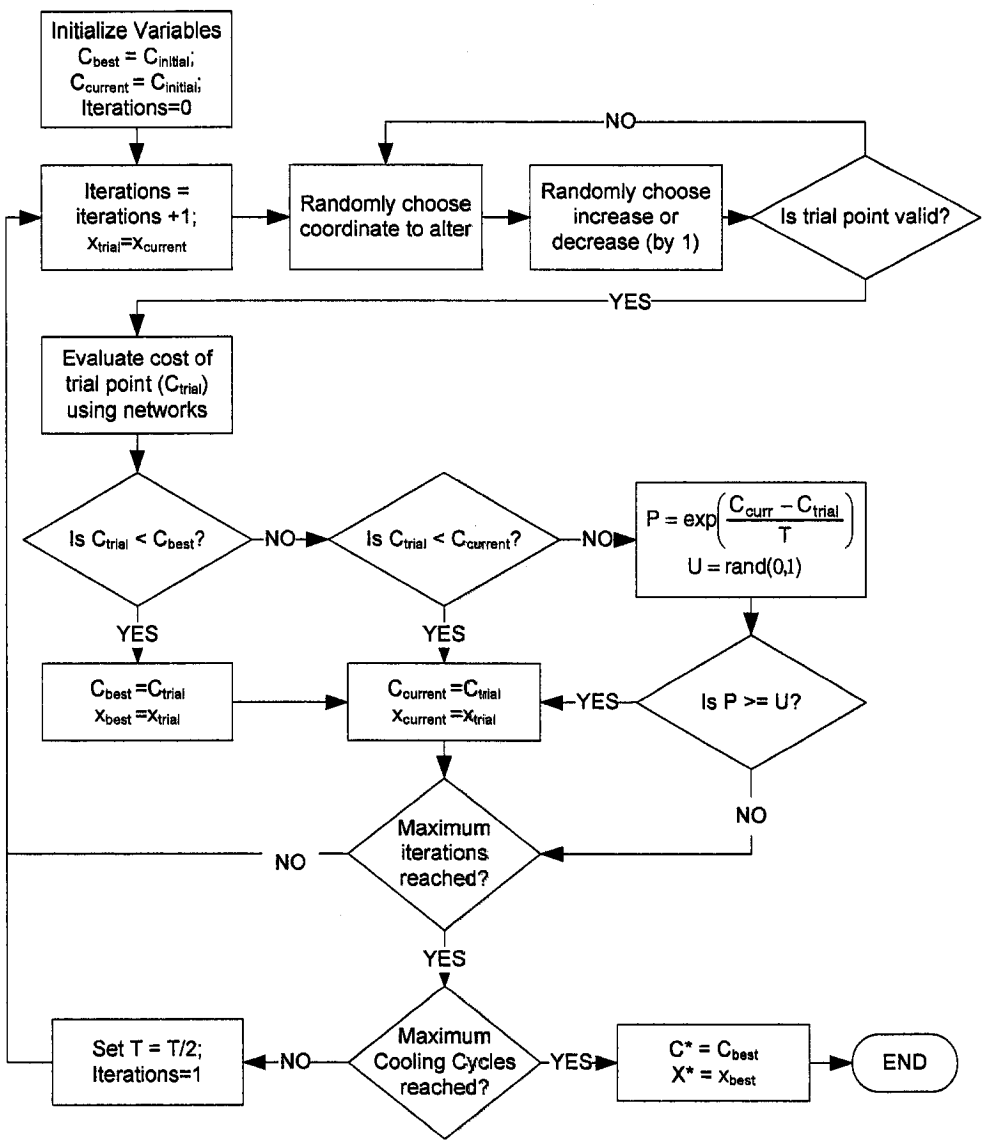

Figure 6.13: Flowchart of Simulated Annealing Algorithm

In this algorithm, the current point and best point are set equal to the initial point. The cost function is evaluated at this point, and the best cost and the current cost are set equal to this value. At each iteration, a new trial point is selected from the neighbourhood of the current point - this neighbourhood is defined as the set of valid design points where only one of the coordinates differs from the current point by +/-1. The function is evaluated at this point, and if the result is lower than the best point seen so far, it is stored as the best point and also as the current point. If it is not the best, but is better than the current point, it is adopted as the current point. If it is not better than the current point, then two values are computed:

$$P = \exp\left(\frac{C_{curr} - C_{trial}}{T}\right)$$

where

C_{curr} = the cost at the current point,
C_{trial} = the cost at the trial point,
T = the cooling temperature.

A random variable is drawn from a uniform (0,1) distribution, and if P is less than this value, the trial point is adopted as the current point. Note that when the temperature, T, is large relative to the difference in cost of the two points, then the value of P will be close to one and the probability of accepting the trial point as the current point is close to one. These iterations continue until the maximum number of iterations for the current cooling cycle is reached, at which point the temperature, T, is reduced by half. As the algorithm progresses, the probability that a non-improving trial point will be accepted as the current point becomes more dependent on the magnitude of the difference between the costs.

For this experiment, the cost function values were set at C_r = \$10/order, C_i = \$10/unit/day, and C_{wip} = \$5/unit/day. The demand rate, D, was 24 orders/day. An initial point was arbitrarily chosen, and the cost at that point was \$137.79. Given the cooling temperature and number of iterations at each temperature, the algorithm performed 45,000 evaluations to arrive at the solution shown in Table 6.17.

Table 6.17: Results of Simulated Annealing Experiment

z_1	k_1	z_2	k_2	z_3	k_3	r_3	z_4	k_4	r_4	Cost
5	2	5	3	1	3	1	1	4	1	$ 103.52

This experiment was repeated a total of 25 times, each with a different starting point. In 24 of the 25 experiments, the resulting solution was the same as in Table 6.17 (see Appendix G for the complete results). The interesting result is that although using the batch parameter will result in a reduction in the amount of setups, the best set of parameters given this cost equation has both batching parameters set to one.

6.7 Developing Exchange Curves

Section 6.6 illustrated that when costs are available, the neural networks may be used to perform optimization. However, when costs are not readily available, which is often an issue in manufacturing, exchange curves provide the ability to understand the trade-offs between performance measures, and the subsequent determination of where the system can operate. In this section, we provide two examples where such exchange curves were constructed.

In the first example, Model A was once again used. In this case, we were interested in identifying the appropriate Kanban strategy for this system. As discussed earlier, Kanban is defined within the PAC system as any combination of PAC parameters where the initial inventory and number of process tags are the same at each station ($k_i = z_i \ \forall i$). Because Model A has a relatively small input space, when we limited eligible points to those that fit the definition of a Kanban system, this was limited even further. Therefore, it was possible to simply evaluate every possible Kanban strategy using the neural network metamodels.

To construct the exchange curve, the two performance measures chosen for the curve were Total Time in System, and Customer Fill Rate. Total Time in System (TT) was a combination of Cycle time (CT) and average finished goods inventory (FGI):

$$TT = CT + \frac{FGI}{D} \qquad [71]$$

where

TT = total time spent in system, including time in finished goods inventory (min),
D = average customer demand rate (units/min).

This measure represents the average total time in the system, including time spent in finished goods inventory. These two measures were plotted for all combinations of points. The resulting graph is shown in Figure 6.14. Note that the minimum fill rate is just over 82%. The networks were trained with a minimum value of four process tags at the first station, and therefore would not be valid for any lower value. According to the definition of Kanban systems, the initial inventory at a cell must equal the number of process tags; therefore, the minimum value of initial inventory at the final store is also four, resulting in a rather high value for fill rate.

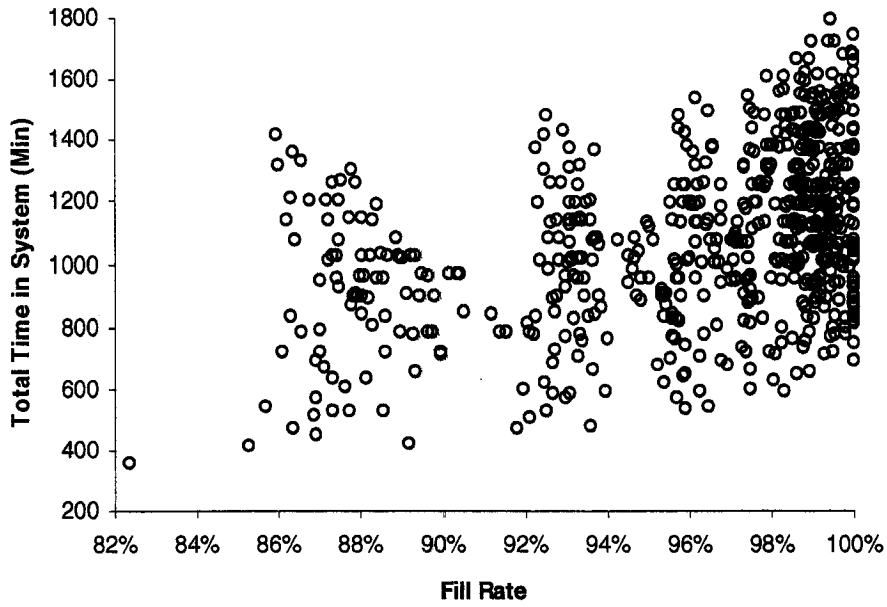

Figure 6.14: Fill Rate vs. Average Time in System for Kanban Strategies, Model A

What may be of interest at this stage are the points which lie at the bottom right of this graph, indicating a fill rate very close to 100% and a lower total time in system. Eliminating all points with fill rates of less than 95% and total time in system greater than 650 minutes resulted in a list of 10 points, listed in Table 6.18. The second point on the list, [7, 3, 1], has a much smaller cycle time than the first point, while only having

slightly smaller fill rate and the same average finished goods inventory level. This list (or an expanded list, by changing the limiting criteria above), presents the decision maker with the ability to make a strategic determination of what to choose.

Table 6.18: Selected Points and Network Results for Kanban Example, Model A

# of Kanbans			Cycle Time	Fill Rate	FG Inv	Total Time
Cell 1	Cell 2	Cell 3				
7	2	2	332.4	0.986	5.3	649.6
7	3	1	277.7	0.983	5.3	594.2
7	2	4	306.4	0.980	5.3	625.9
8	2	1	232.4	0.975	6.1	599.6
7	2	1	230.8	0.964	5.1	537.8
6	2	2	330.7	0.962	4.3	589.0
6	3	1	275.2	0.959	4.3	533.1
6	4	4	377.6	0.958	4.4	642.2
6	2	4	304.7	0.957	4.4	566.6
6	6	1	360.0	0.954	4.4	621.1

To construct an exchange curve, we opted to plot the percentage of demands not met from stock (1 – Fill Rate). Figure 6.15 contains a subset of the same data used for Figure 6.14, but only includes points with a total time in system of less than 800 minutes (since we can achieve a 100% fill rate with less than 700 minutes in the system).

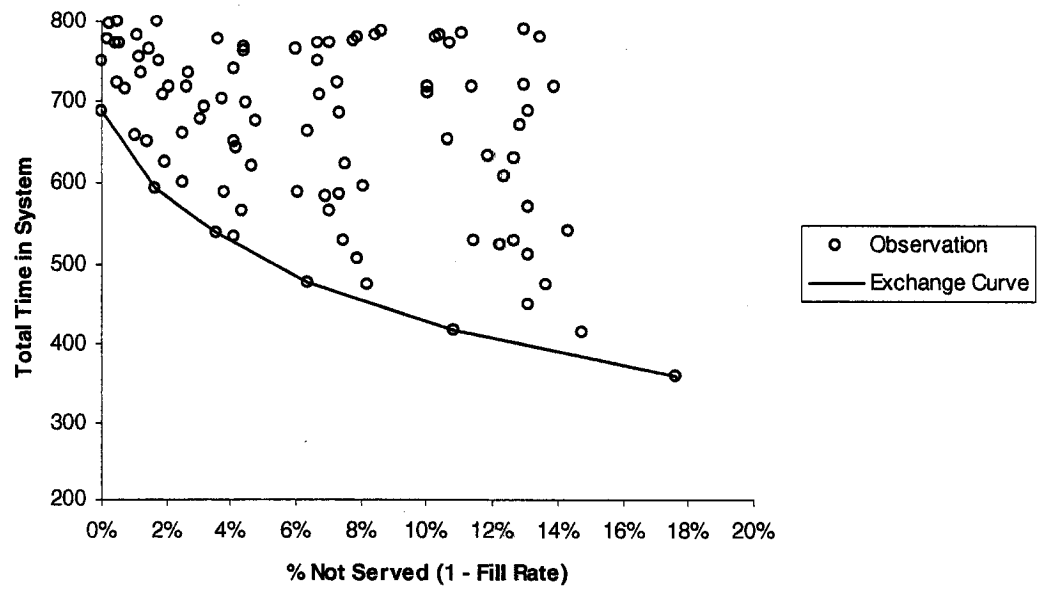

Figure 6.15: Exchange Curve for Kanban Strategies, Model A

A second optimal policy curve was developed for this model by not constraining the strategy to Kanban, but by letting the PAC parameters take on any eligible value. The optimal policy curve was once again determined and the result plotted against the Kanban curve (Figure 6.16) and labelled as "Hybrid". As can be seen from this curve, by not limiting the control strategy to a Kanban strategy, significant reductions in total time in system (and thus inventory) as well as gains in customer service can be achieved. Bonvik, Couch and Gershwin (1997) conducted a similar study to compare inventory levels and customer service for Kanban strategies and hybrid strategies for a four-station serial production line. They constructed the optimal policy curves for this system by conducting 50-100 simulations at 1250 different Kanban strategies and 3000 hybrid strategies. The computational expense of constructing such optimal policy curves using the neural networks to evaluate each strategy is significantly lower for problems of this magnitude.

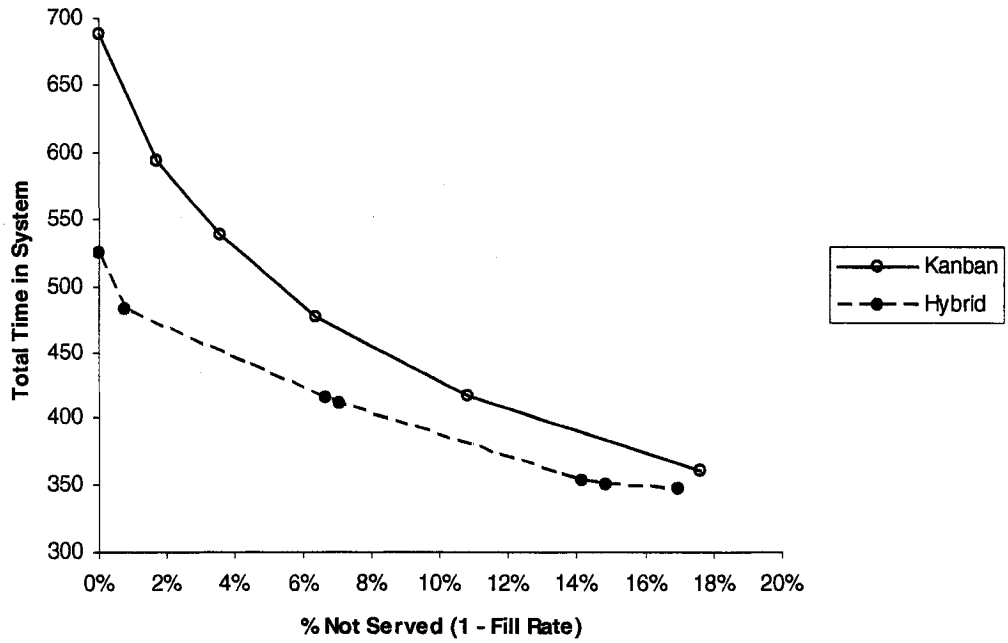

Figure 6.16: Exchange Curves for Kanban and Hybrid Strategies, Model A

When models are more complex and the input space is larger, this approach for developing the exchange curve is not attractive. For example, with Model C, there are

over 800 million eligible parameter combinations. Thus, an optimization heuristic such as simulated annealing will be necessary to construct the curve. To illustrate this, we used the same simulated annealing algorithm discussed in Section 6.6, along with the networks previously trained for Model C.

The two measures we opted to study for this system were the percentage of customer demands not immediately satisfied from finished goods, and waiting inventory. This system produces two products; 40% of customer demand is for Product 1, and 60% is for Product 2. Therefore, for the first measure, the percentage of demands not immediately filled from inventory, 1 - $\bar{f}$, was calculated as follows:

$$1-\bar{f} = 1-(0.40v_3 + 0.60v_4) \qquad [72]$$

where

$\bar{f}$ = average fill rate for both products
v_3 = fill rate for product 1
v_4 = fill rate for product 2

The second measure is the total of the waiting inventory in the system, I_w. Since the measures for WIP do not include material in process at a cell, the sum of the WIP inventory and finished goods inventory will be the total number of products waiting within the system. Although this ignores the value added by each processing step, it is still proportional to the average amount of raw materials tied up in the system. Therefore, I_w is defined as

$$I_w = (v_5 + v_6) + (v_7 + v_8 + v_9) \qquad [73]$$

where

$(v_5 + v_6)$= sum of finished goods inventory (both products), and
$(v_7 + v_8 + v_9)$= sum of work-in-process inventory at processing cells

To construct the curve, we first constructed the following function, C, as follows:

$$C = \theta(1-\bar{f}) + (1-\theta)I'_w \qquad [74]$$

where

θ = a weighting parameter, and

$$I'_w = \frac{I_w - I_w^{\min}}{I_w^{\max} - I_w^{\min}}$$

where

$I_w^{\min}$ = minimum observation for I_w from the training set data, and

$I_w^{\max}$ = maximum observation for I_w from the training set data.

The reason for the calculation of I'_w was that all values of average fill rate, $\bar{f}$, were between 0 and 1, so therefore I_w was roughly "normalized" so that it would be closer in magnitude to values of $\bar{f}$.

Starting at $\theta = 0.005$ (rather than zero, to ensure there was still some weight placed on the fill rate function), this function was minimized using the simulated annealing algorithm discussed above. The value of θ was then increased to 0.05, and the optimization repeated. The value of θ was repeatedly incremented in steps of 0.05, with an optimization applied at each stage, until θ reached 0.95. Finally, one minimization, with of $\theta = 0.995$, was carried out. Thus, a total of 21 optimizations were carried out, with five repetitions of the simulated annealing algorithm, each with a different initial point, for each value of θ. The points and resulting performance measures were then plotted, and the convex hull of these points determined. The curve is shown in Figure 6.17 .

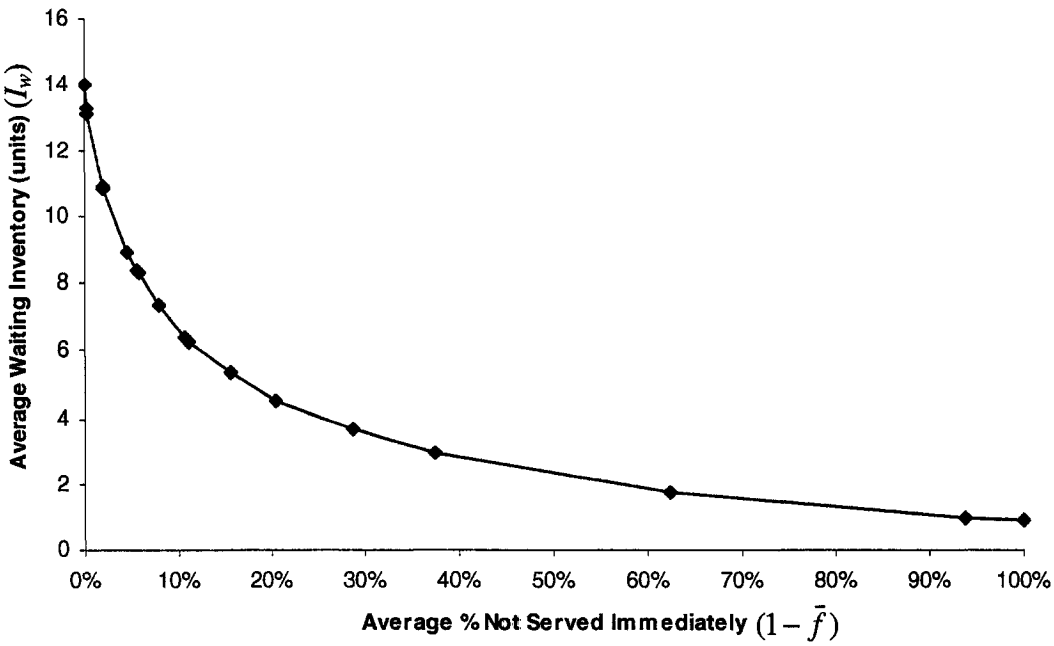

Figure 6.17: Exchange Curve for Model C

Again, this curve represents the optimal policy curve for the example system. As can be seen, only small increases in WIP are required to improve fill rate dramatically up to about 50%, at which time the curve begins to increase steeply.

6.8 Accuracy, Generalization, and Comparison with Regression Models

In this section, we demonstrate that neural networks are a good choice for the metamodeling approach to this problem. A more complex model, Model D, was used for the experiments described in this section. Model D has four processing stations, where two of them are assembly stations (Figure 6.18). Some products are required in quantities of more than one. Again, customer demand arrives according to a Poisson process with mean time between arrivals of 60 minutes. Because there are no cells which produce more than one product, batch sizes of one were assumed for all products. Some of the materials require travel time to arrive at the requesting cell. As well, Cell 1 has

two parallel machines, which both produce Product 4. The system parameters for Model D are shown in Table 6.19.

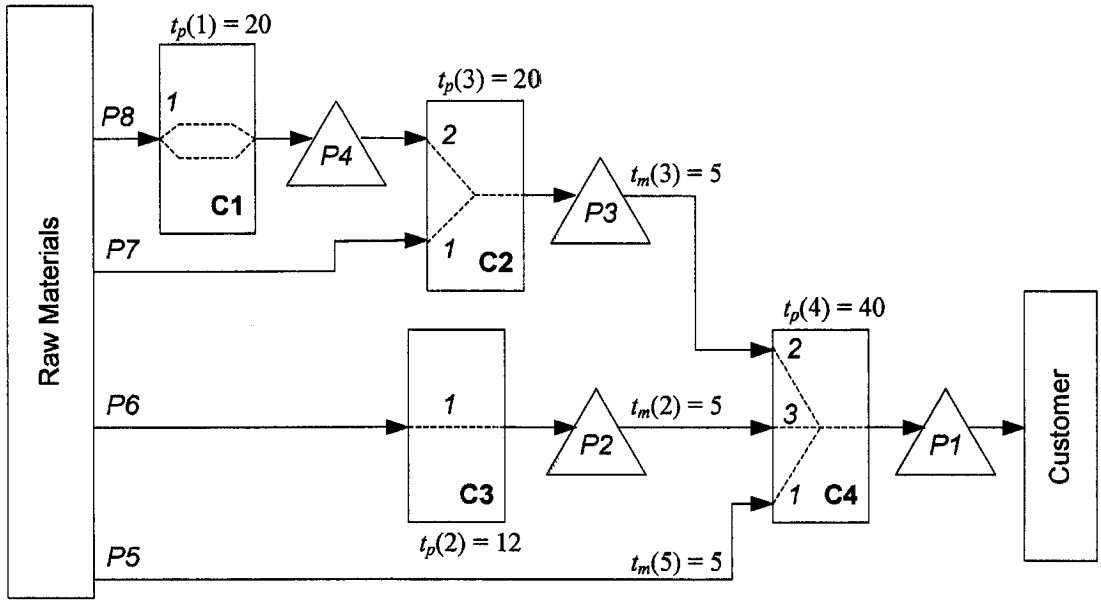

Figure 6.18: Model D

Table 6.19: System Parameters for Model D

Products	Station	Inputs (Part and Qty)
1	4	2 (Qty 1), 3 (Qty 2), 5 (Qty 1)
2	3	6 (Qty 1)
3	2	4 (Qty 2), 7 (Qty 1)
4	1	8 (Qty 1)

Where some cells require parts in quantities greater than one in order to produce one product, the issuance of one PA card for such a product will generate multiple orders for component parts. For example, when a PA card for Product 1 is generated and sent to Cell 4, the cell will generate two orders for Product 3, three orders for Product 2, and one order for Product 5.

Rules 1 and 2 were applicable to this model, resulting in the following constraints:

$$\text{Rule 1: } k_2 \leq 3k_1 + z_2$$

$$k_3 \leq 2k_1 + z_3$$

$$k_4 \leq 2k_3 + z_4$$

Rule 2: $k_4 \geq 2$

Given these rules and the minimum and maximum allowable values for the parameters, the input space was calculated to include 3,457,938 valid points. The ranges for each parameter were divided into three levels (Appendix H), and a single point was generated for each valid level combination, resulting in a training set of 6563 points. A total of ten neural networks were trained using this dataset (Figure 6.19). The results of the training can be found in Appendix H.

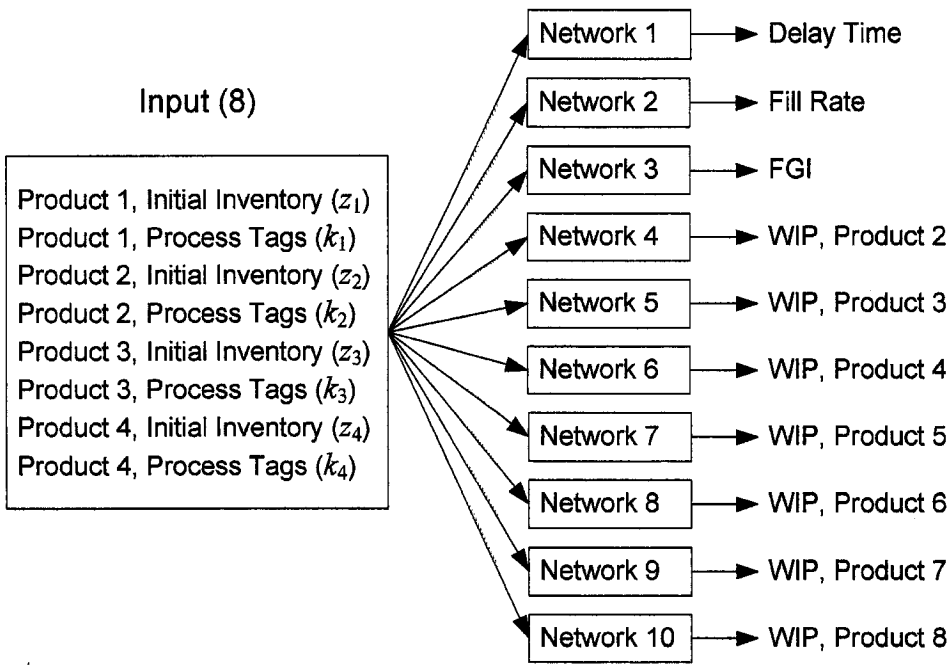

Figure 6.19: Neural Networks for Model D

Once again, three points not included in the dataset were chosen and evaluated using the networks, and as well, each was simulated 20 times. A 99% confidence interval for the mean of each performance measure was also calculated from the simulation runs. Table 6.20 shows the results of those comparisons. While some of the neural network

results do not fall within the 99% confidence intervals for the mean value, those that fail are very close to what amounts to quite narrow confidence intervals.

Table 6.20: Results for Sample Points, Model D

Design Point 1

		CI for Simulations	Network Response
$z_1 = 4$	Delay	(18.0, 22.6)	20.1
$k_1 = 6$	Fill Rate	(0.761, 0.786)	0.770
$z_2 = 3$	FGI	(2.03, 2.11)	2.04
$k_2 = 1$	WIP 2	(4.63, 4.86)	4.90
$z_3 = 2$	WIP 3	(1.54, 1.59)	1.59
$k_3 = 4$	WIP 4	(1.93, 2.00)	1.96
$z_4 = 2$	WIP 5	(1.35, 1.45)	1.42
$k_4 = 4$	WIP 6	(0.00, 0.00)	0.00
	WIP 7	(1.61, 1.67)	1.66
	WIP 8	(0.95, 0.97)	0.94

Design Point 2

		CI for Simulations	Network Response
$z_1 = 0$	Delay	(206.1, 213.9)	228.3
$k_1 = 8$	Fill Rate	(0.000, 0.000)	0.004
$z_2 = 0$	FGI	(0.00, 0.00)	0.02
$k_2 = 1$	WIP 2	(5.28, 5.58)	5.48
$z_3 = 0$	WIP 3	(1.08, 1.17)	1.09
$k_3 = 4$	WIP 4	(1.21, 1.25)	1.26
$z_4 = 0$	WIP 5	(2.60, 2.71)	2.64
$k_4 = 4$	WIP 6	(0.00, 0.00)	0.00
	WIP 7	(2.04, 2.08)	2.06
	WIP 8	(0.86, 0.88)	0.83

Design Point 3

		CI for Simulations	Network Response
$z_1 = 6$	Delay	(3.6, 5.6)	4.9
$k_1 = 5$	Fill Rate	(0.936, 0.950)	0.944
$z_2 = 6$	FGI	(4.17, 4.25)	4.27
$k_2 = 3$	WIP 2	(6.41, 6.57)	6.44
$z_3 = 4$	WIP 3	(2.74, 2.80)	2.72
$k_3 = 6$	WIP 4	(2.14, 2.22)	2.20
$z_4 = 2$	WIP 5	(0.95, 1.01)	0.97
$k_4 = 6$	WIP 6	(0.91, 0.92)	0.91
	WIP 7	(2.09, 2.15)	2.12
	WIP 8	(1.68, 1.71)	1.68

6.8.1 Generalization

To illustrate the generalization properties of neural networks, a second set of 6,561 input data points was generated using the same rules as were used to generate the training set. Each point was tested to ensure that it did not appear in the training dataset. Then, all data points in this new set were evaluated using the ten neural networks produced for Model D. A comparison of the error results for the training set and the test set are shown in Table 6.21. As can be seen from this data, the overall error results for the test data set are quite comparable to the training data set, even though a cross-validation procedure was not used.

Table 6.21: Error Results for Training Data Set and Test Data Set, Model D

	Observations		MSE		Mean Error		Mean Absolute Error	
Network	Min	Max	Training Set	Test Set	Training Set	Test Set	Training Set	Test Set
C. Delay	0	275.46	10.76700	13.21200	0.01688	-0.06573	2.2076	2.3447
Fill Rate	0	1	0.00023	0.00026	0.00002	0.00042	0.0109	0.0115
FGI	0	8.634	0.00545	0.00610	-0.00114	0.00124	0.0539	0.0564
WIP 2	1.446	16.591	0.02686	0.02898	0.00073	-0.00149	0.1257	0.1302
WIP 3	0.736	10.646	0.00772	0.00854	0.00001	-0.00233	0.0692	0.0720
WIP 4	0.88	15.27	0.01070	0.01100	0.00001	0.00128	0.0809	0.0815
WIP 5	0.502	2.861	0.00558	0.00592	0.00019	0.00067	0.0573	0.0590
WIP 6	0	2.125	0.00205	0.00211	-0.00001	-0.00016	0.0315	0.0319
WIP 7	0.878	3.763	0.00836	0.01558	-0.00003	-0.01218	0.0683	0.0802
WIP 8	0	3.008	0.00341	0.00399	-0.00001	-0.00110	0.0404	0.0425

The post regression plots for the customer delay time for both the training dataset and the test dataset are shown in Figure 6.20 and Figure 6.21. The plots for the other networks are shown in Appendix H.

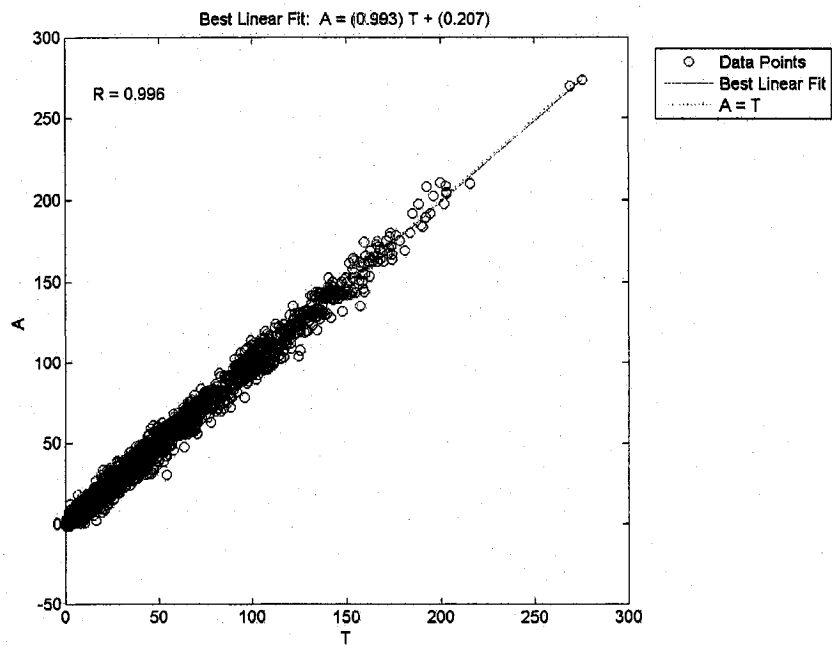

Figure 6.20: Simulation Output vs. Neural Network Output, Customer Delay, Model D

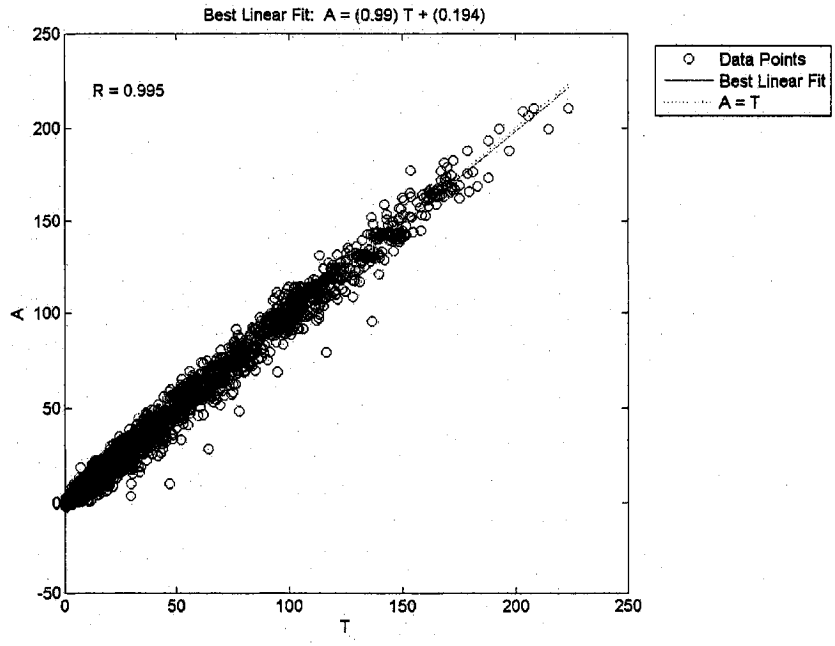

Figure 6.21: Simulation Output vs. Neural Network Output, Customer Delay, Test Dataset, Model D

6.8.2 The Networks Produce the Expected Value Functions

It has been assumed that the networks produce unbiased estimators of the expected value of the simulation response. In order to demonstrate the validity of this assumption, an additional test was conducted using the neural networks trained for Model D.

If the network does in fact produce unbiased estimators, then, over repeated experiments, the average difference between the simulation response and the network prediction should equal zero.

$$E[v(x) - s(x)] = 0 \qquad [75]$$

where

$v(x)$ = the network output for input x,
$s(x)$ = a random simulation response for input x.

As described in Section 6.1.3, the ranges of values for each parameter were divided into levels. In this experiment, ten unique points from each valid combination of levels were randomly chosen. For example, one valid level combination is shown in Table 6.22. In the training set, one point would be generated for this level combination by randomly selecting a value from each range for each parameter. For this experiment, 10 such points were randomly generated. Because there were 6561 valid combinations of levels for Model D input, this resulted in 65,610 design points.

Table 6.22: Example of a Valid Level Combination for Model D

	Level	Range
z_1	Low	0, 3
k_1	High	9, 12
z_2	Mid	4, 8
k_2	Mid	4, 6
z_3	Mid	4, 6
k_3	Mid	7, 10
z_4	Low	0, 5
k_4	Low	2, 4

All of the points in this dataset were simulated once using the PACSIM model and the performance measures were recorded. The points were also passed through each of the neural networks to obtain the performance measures estimates.

Once the results were obtained from both the simulation and the networks, each set of design points (10 points from the same region) were compared. The difference between the neural network result and the simulation result was calculated, and then, for each set of ten points, the average of these differences was calculated. A 99% confidence interval ($t_{9,0.99} = 3.250$) for the mean difference for each set was calculated, therefore resulting in 6561 confidence intervals for each network. For each of the networks, the number of confidence intervals which contained zero was counted, and the results are shown in Table 6.23. For all the sets of 10 points tested for each network, the confidence interval contained zero at least 96.7% of the time.

Table 6.23: Number of Confidence Intervals for Level Sets which Contain Zero

	Intervals Containing Zero	
Network	Number	Percentage
1	6342	96.7%
2	6405	97.6%
3	6443	98.2%
4	6463	98.5%
5	6439	98.1%
6	6459	98.4%
7	6452	98.3%
8	6407	97.7%
9	6430	98.0%
10	6453	98.4%

The results from this experiment support the claim that the trained networks are satisfactorily unbiased in the prediction of the simulation response. For at least 96.7% of the regions, the hypothesis could not be rejected at the 99% level (which is about 3σ, the same amount used in control charting).

6.8.3 Comparison with Regression Models

In this experiment, second and third order regression models with two-way interactions were fit to the same data used to train the neural networks for the eight input

parameter Model D. The forms for the models are shown in Equations [76] and [77]. The coefficients were calculated using Matlab.

Second Order Model:

$$y = a_0 + \sum_{i=1}^{8} a_i p_i + \sum_{i=1}^{8} b_i p_i^2 + \sum_{i=1}^{7} \sum_{j=i+1}^{8} d_{ij} p_i p_j \qquad [76]$$

Third Order Model:

$$y = a_0 + \sum_{i=1}^{8} a_i p_i + \sum_{i=1}^{8} b_i p_i^2 + \sum_{i=1}^{8} c_i p_i^3 + \sum_{i=1}^{7} \sum_{j=i+1}^{8} d_{ij} p_i p_j \qquad [77]$$

where

y = regression value (estimate of output)
a_i, b_i, c_i, d_{ij} = regression coefficients
p_i = input values (PAC parameters)

The second order model contained a total of 45 terms, and the third order model contained 53 terms. The test data discussed in Section 6.8.1 was also presented to the regression models, and the mean squared error (MSE) and the mean absolute error (MAE) were calculated for both. Finally, for the two regression models, for each performance measure, the order of difference between each regression model and the neural network models were calculated by dividing the MSE of the regression model by the MSE of the neural network model. The MSE results are presented in Table 6.24.

Table 6.24: Neural Network and Regression Results for Mean Squared Error (MSE), Model D

	Neural Networks		2nd Order Regression			3rd Order Regression		
Network:	Training	Test	Training	Test	Factor	Training	Test	Factor
1 (Delay)	10.767	13.212	99.155	101.41	7.7	46.884	48.31	3.7
2 (Fill Rate)	0.0002	0.0003	0.0021	0.0023	8.6	0.0012	0.0012	4.7
3 (FGI)	0.0054	0.0061	0.0330	0.0345	5.7	0.0165	0.0179	2.9
4 (WIP 2)	0.0269	0.0290	0.1209	0.1307	4.5	0.0713	0.0788	2.7
5 (WIP 3)	0.0077	0.0085	0.0517	0.0572	6.7	0.0268	0.0303	3.6
6 (WIP 4)	0.0107	0.0110	0.0515	0.0517	4.7	0.0154	0.0160	1.5
7 (WIP 5)	0.0056	0.0059	0.0162	0.0174	2.9	0.0107	0.0116	2.0
8 (WIP 6)	0.0021	0.0021	0.0060	0.0066	3.1	0.0035	0.0039	1.8
9 (WIP 7)	0.0084	0.0156	0.0217	0.0244	1.6	0.0126	0.0152	1.0
10 (WIP 8)	0.0034	0.0040	0.0116	0.0164	4.1	0.0095	0.0138	3.5

As can be seen in the table, the third order regression model outperformed the second order model in each case, as would be expected. In comparison to the neural network models, the third order regression model achieved virtually the same MSE for the WIP value for Product 7 (WIP7), but resulted in an MSE about 4.7 times higher than the networks for the Fill Rate. Closer examination of the post-regression plots for both the customer service rate and fill rate indicates that the third order model is not sufficient for modeling these two performance measures, especially when compared with the neural network result.

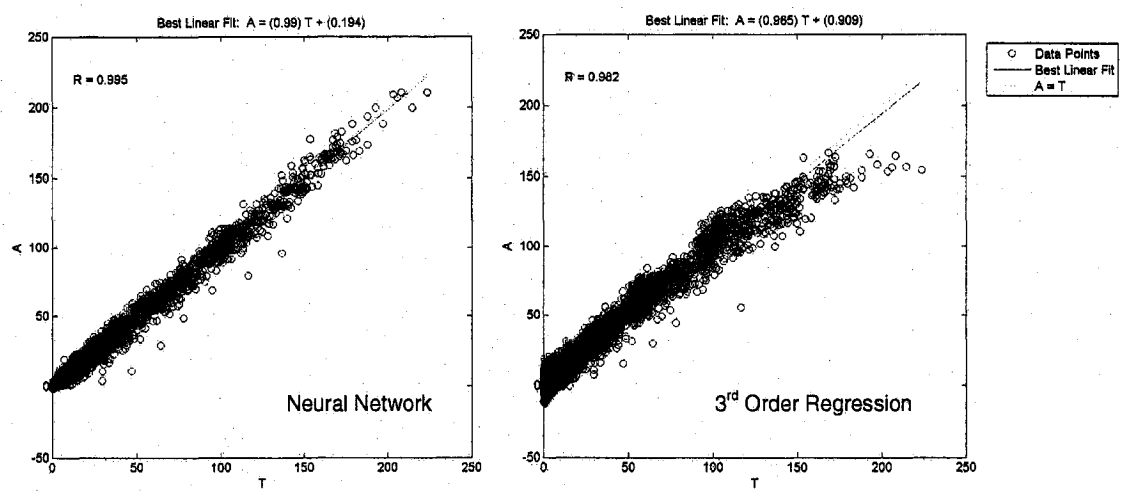

Figure 6.22: Comparison of Post Regression Plots, Test Set, Customer Delay, Model D

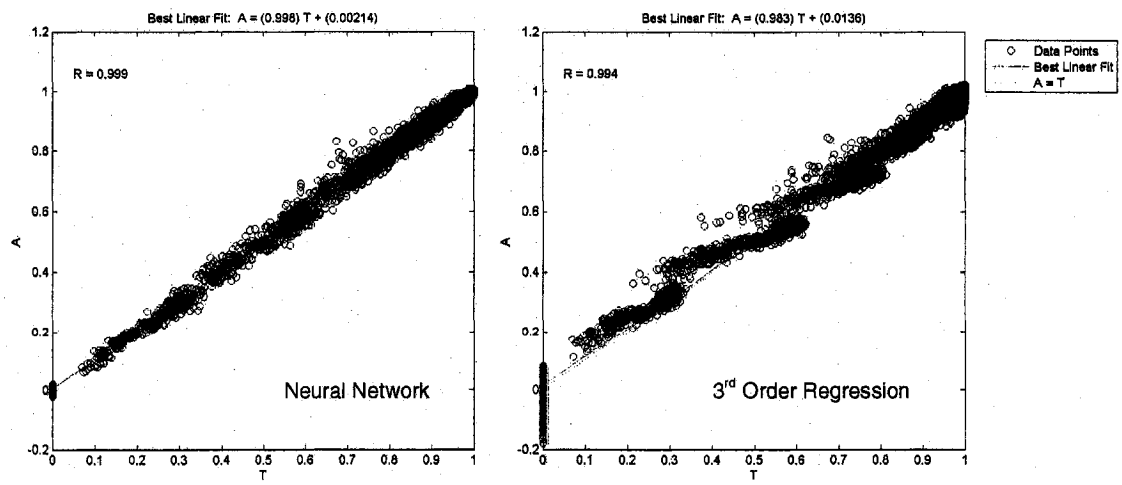

Figure 6.23: Comparison of Post Regression Plots, Test Set, Fill Rate, Model D

More experiments could be conducted with models with higher-order interactions, or higher-order powers, until a better regression model was found. Obviously the third order regression model is as accurate as the neural network model for WIP7 and much easier to work with. However, none of the other results achieved this level of accuracy. Obvoiusly if we know that some particular form is appropriate, such as models with terms such as $1/p_i$ we should be able to get good fits with regression models. However, the purpose of this framework is to enable the study of any complex system, and we are not likely, *a priori*, to have good ideas for specific appropriate model forms. Fully populated third order models will have 149 terms, a number close to the number of weights in the neural networks. This experiment demonstrates that simple fully populated quadratic models and the extension of these models with third order monomial terms are insufficient for the purposes of our framework. This supports the notion that neural networks are the more appropriate choice.

6.9 Concluding Remarks on Experiments

As can be seen from the preceding examples, neural networks can be trained to be are reasonably accurate, unbiased estimators of the expected value functions for system performance. The network functions provide easy and flexible approaches to analyzing or optimizing the system responses when compared to the alternatives of direct simulation or simulation optimization.

Our experiments have also indicated that the results coming out of the neural network analysis can be reasonably accurate portrayals of the simulation expected values. For points not included in the original training set, and therefore not part of the fitting process, we demonstrated that the neural networks provided reasonable approximations of the expected values. We showed that the neural network results were often contained within a confidence interval constructed for the true mean, and therefore we could not reject the hypothesis that the network was in fact producing the expected values; where this was not true, we showed that the relative error between the network approximation and the true expected value were tightly bounded.

We demonstrated that this framework provides the means to determine the most appropriate control strategy for a manufacturing system. We demonstrated that exchange curves can be constructed, even for complex models, to allow for the determination of the right combination of performance measures for the system. We showed how exchange curves for different types of strategies can be constructed and compared, with much lower computational requirements than using simulation alone. We also demonstrated the framework may be used to minimize a cost function of the performance measures, or to optimize one performance function subject to constraints on the others. Therefore, this framework addresses the original problem posed by Buzacott and Shanthikumar.

CHAPTER 7

OBSERVATIONS, CONCLUSIONS, AND OPPORTUNITIES FOR FUTURE WORK

7.1 Summary

We have addressed the important problem posed by Buzacott and Shanthikumar (1992) of developing a framework for the systematic comparison of various manufacturing control schemes and selection of the most appropriate control strategy for a manufacturing system, provides a unifying formalism for a broad range of control strategies, including the conventional strategies such as Kanban, CONWIP, and Base Stock, to name a few. Since alternate control strategies arise from the PAC system as a result of choices of PAC parameters, the natural questions to ask are how does system performance vary as these parameter choices vary and how can we optimize the choice of parameters for a given manufacturing setting. The contribution of this thesis is the development of a framework that makes it possible to study these questions.

Since complex systems cannot be modeled analytically, a simulation model of a system operating under the PAC scheme was required. The PACSIM simulation model presented in this thesis is a result of ongoing work at Dalhousie. It provides estimates of performance measures, such as average inventory at each production cell and customer service levels for each finished product, for systems with complexities such as multiple-product machines requiring setup time, travel times for requisitioned materials, assembly stations requiring multiple subassemblies, and systems which produce more than one finished product. Because PACSIM was developed in a high-level language (FORTRAN), it can achieve execution speeds significantly higher than models built in standard simulation packages.

We established in this thesis that the appropriate approach to this problem was to provide a flexible means of analysis for the determination of the right operating strategy. The definition of the best strategy is often difficult to determine in light of the conflicting

performance measures of the system, and the difficulty in determining the various costs, such as holding costs and lost order costs. Although the use of a simulation model for performance estimation suggested the use of a simulation optimization technique, we argued that this was not the right approach for this problem. Such techniques are computationally difficult to apply to large problems, and in some cases, are inappropriate as a modeling concept. We demonstrated that simulation metamodeling is the appropriate choice for our framework. In metamodeling, the goal is to approximate the expected value function for each performance measure of interest, with respect to the PAC scheme parameters. Such metamodels provide the ability to apply deterministic optimization techniques, as well as the ability to explore trade-offs amongst the various performance measures, so that the best control strategy may be selected based on policies and outside decision factors not easily integrated into any optimization approach.

Of the metamodel approaches available, neural networks presented many attractive advantages. In this thesis, we demonstrated that feedforward neural networks are a highly practical and feasible approach, even for large numbers of input parameters. These models can be constructed easily based on a subset of data sampled from the input space. Our simulation experimental design displayed good space-filling properties, and was a contributing factor to the construction of such accurate metamodels. The rules we developed for the selection of PAC parameter combinations ensured that only feasible and reasonable design points were selected for training the network metamodels, and were an important contribution.

Finally, through experiments conducted on several different manufacturing settings, we provided the details on the implementation of this framework, and also provided examples of the various types of analysis possible with this framework. We therefore demonstrated that this framework addresses the original problem posed by Buzacott and Shanthikumar.

7.2 Areas for Further Research

7.2.1 Modeling MRP Strategies

The examples used in this work did not include forecasting, and therefore MRP type systems which release work into the system based on forecasted future demand were not represented. However, with only some modifications to the original simulation model, MRP type systems could easily be included by introducing a forecast into the model, and generating customer orders at some time previous to the expected arrival of this forecasted demand. In fact, Bielunska-Perlikowski (1997) did originally create such a model. However, she assumed (as did Buzacott and Shanthikumar, 1992), that forecasts were perfect and therefore the only measurement of interest was the system's ability to meet demand given this advance notice. In the PAC scheme, MRP type systems are modeled by removing the limit on the number of process tags at any station, therefore allowing orders (and information) to flow freely upstream each time an order for a finished product is received by the system. Because any study of MRP system would not be complete without examining the effects of forecasting errors, cancellation notices and surplus tags would also have to be included in the simulation.

This then brings up at least two opportunities for future work. The first would involve determining a method to fairly compare MRP type systems using the PAC scheme, including imperfect forecasts, the use of the delay parameter, τ_i, with pull type systems where the delay parameter is not employed and the system typically responds to actual demand. The second would involve the analysis of an MRP system specifically, and whether or not such a system would see an improvement in performance if limits on the number of available process tags were placed on some or all of the production cells. One criticism of MRP is that jobs are released into the system without regard for the number of jobs already in the system (Hopp and Spearman, 2001); therefore, a limit on information flow could reduce average work in process inventory and provide a lower, and perhaps less variable, average lead time for jobs. Both of these issues should be investigated.

7.2.2 Priority Schemes

In systems where a single cell produces more than one product, the priority scheme used to determine the sequence in which PA cards waiting in the queue are processed was assumed to be a management decision made in advance. This was also the case where a single store supplied a part to more than one downstream cell. As was discussed earlier, the priority scheme used will affect the performance of the system. Our assumption that a FIFO strategy was used to determine the next product to produce at a multi-product cell could have resulted in a large number of changeovers. Although not dealt with here, there will be situations where the capacity of the cell would be insufficient to keep up with demand if such a strategy were employed, especially if the cell were the bottleneck in the process. However, a strategy designed to reduce the number of changeovers at a cell may allow the cell to produce at the desired level, albeit at the expense of an increase in inventory, as longer production runs of a each product would be necessary. The batching parameter in the PAC system, r_i, can influence this to some degree, since, with the use of a FIFO system, a batch of r_i orders for a single product arriving at a cell would result in the cell would producing r_i products in a row before changing to another product. However, there may be opportunities to improve overall system performance by the selection of a different type of priority scheme at any of these cells or store. Therefore, the type of priority rule could be a parameter value for each cell/store where such decisions are relevant. Various types of priority schemes (or queue disciplines), such as shortest processing time (SPT), cμ-rule (Rosa-Hatko and Gunn, 1997), or perhaps simply processing all authorizations for the same product currently in the queue until none remain or a time limit is reached. A set list of priority schemes could be coded in PACSIM, and then an input parameter would instruct the program on which type to use. Including this input parameter in the neural network metamodels would be more of a challenge. The other PAC parameters are real-valued input; the selection of a strategy is not. Further work would be necessary to determine the best neural network architecture to deal with this type of input.

7.2.3 Alternative Performance Measures

While PACSIM provides several common performance measurements, there are other measures which should be explored. One such measurement of interest would be the maximum possible throughput of the system, given different PAC scheme parameters, and the corresponding average inventory levels required to achieve that throughput. This would require a change to the simulation model so that the system no longer responds to arriving demand, but instead to the completion of a product. One possible way to model this would be to generate an order and requisition at the start of the simulation for each unit of the initial inventory (z) assigned to the finished products store, and then, each time a finished product is completed and arrives at this store, another "customer" order and requisition would be generated for the product in order to immediately remove it from the store. More work should be done to explore this option.

7.2.4 Modeling Other Real-World System Complexities in PACSIM

While PACSIM does allow for some realistic complexities, further work would need to be done on this model to allow increase the ability of PACSIM to accurately model actual systems. For example, it is assumed in PACSIM that items are free to move individually; once a product is completed at a cell, it immediately begins travel to the store, and will immediately begin travel from that store to a cell once a requisition is received. In reality, there may be some requirement for parts to travel in batches, due to limitations on transportation equipment. Although PA cards may be transmitted in batches, to enable the sequential processing of like components at a cell, it is assumed in the model that processing takes place sequentially until all the PA cards have been completed. However, in some cases, such as a heat treating operation which takes several hours, batch processing may be employed, and all PA cards would be processed and released at the same time. It is assumed that the machines in the cells are always available; in reality, breakdowns and operator absences will affect the performance of the system. The PACSIM model again could be modified to include such random breakdowns or stoppages. Another modification would permit sequence dependent setup up times at multi-product production cells. Finally, it is also assumed although the time

between demand arrivals is random, the quantity of the demand is always one; this should be modified to include not only random demand arrival times, but also random amounts, which is often a characteristic of many real systems.

It is important that this framework be applied in a real manufacturing environment; however, the modifications discussed in this section should be made to PACSIM prior to such an application.

7.2.5 Neural Network Development

We demonstrated that neural network metamodels are a useful tool for analysis and optimization of manufacturing systems operating under a PAC scheme. Our focus was not to determine the best possible network architecture, but to simply demonstrate that a reasonably accurate neural network could easily constructed for such purposes. Therefore, the design and training of the networks was achieved primarily through trial and error.

The MATLAB Neural Network Toolbox was chosen for network training because it was readily available, was easy to use, and worked well. For each network, a small number of hidden nodes were arbitrarily chosen, and then the network was trained. This number was increased until the resulting MSE seemed reasonable, and only small improvements could be made by adding more nodes. The post-regression plots were visually examined for obvious bias. For most examples, there were sufficient points to reasonably guarantee good generalization as the number of points well exceeded the number of network weights, however cross-validation is one technique which could have been applied to improve the results. There are statistical validation techniques in the literature for choosing a model architecture (e.g. Anders and Korn, 1999). However, we simply chose the single hidden layer, using the logistic function for the transfer function, and a linear output layer due to their ease of use. Other architectures or different transfer functions could be investigated, however we found that choice of architecture did provide a reasonable degree of accuracy.

Although we did have some knowledge as to the relationship amongst the performance measurements, we did not incorporate any of this knowledge into the design

of the networks. The networks are continuous functions, and the networks used in these examples were designed with a linear output layer. Therefore, negative output values were possible, and were sometimes observed, even though negative measures were not possible in this model. As well, there could have been some knowledge built into these networks – for example, where the initial inventory at a finished goods stock point were set to zero, the fill rate and average finished goods inventory measures will always be zero. This knowledge was not built into the models, and therefore these measures were rarely exactly zero.

7.3 Concluding Remarks

In closing, the framework presented here enables the steady state performance analysis of complex manufacturing systems under a variety of traditional manufacturing control schemes, hybrids of these schemes, and any other scheme that can be described using the PAC parameters. As such, the framework presents important opportunities for future work as discussed above.

References

1. Akturk, M.S. and Erhun, F. (1999). An Overview of Design and Operational Issues of Kanban Systems. *International Journal of Production Research*, 37(17), 3859-3881.

2. Alam, F.M., McNaught, K.R., and Ringrose, T.J. (2004). A Comparison of Experimental Designs in the Development of a Neural Network Simulation Metamodel. *Simulation Modelling Practice and Theory*, 12: 559-578.

3. Alrefaei, M.H. and Andradottir, S. (1997). Accelerating the Convergence of the Stochastic Ruler Method for Discrete Stochastic Optimization. *Proceedings of the 1997 Winter Simulation Conference, S. Andradottir, K.J. Healy, D.H. Withers and B.L. Nelson, eds.*, 352-357.

4. Alrefaei, M.H. and Andradottir, S. (1999). A Simulated Annealing Algorithm with Constant Temperature for Discrete Stochastic Optimization. *Management Science*, 45(5): 748-764.

5. Altiok, T. and Stidham, S. (1983). The Allocation of Interstage Buffer Capacities in Production Lines. *IIE Transactions*, 15(4):292-299.

6. Anders, U. and Korn, O. (1999). Model Selection in Neural Networks. *Neural Networks*, 12: 309-323.

7. Andradottir, S. (1992). Discrete Optimization in Simulation: A Method and Applications. *Proceedings of the 1992 Winter Simulation Conference, J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, eds.*, 483-486.

8. Andradottir, S. (1995a). A Method for Discrete Stochastic Optimization. *Management Science*, 41: 1946-1961.

9. Andradottir, S. (1995b). A Stochastic Approximation Algorithm with Varying Bounds. Operations Research, 1037-1048.

10. Andradottir, S. (1996). A Global Search Method for Discrete Stochastic Optimization. *SIAM Journal on Optimization*, 6(2): 513-530.

11. Andradottir, S. (1998). A Review of Simulation Optimization Techniques. *Proceedings of the 1998 Winter Simulation Conference, D.J. Medeiros, E.F. Watson, J.S. Carson, and M.S. Manivannan, eds.*, 151-158.

12. Askin, R.G. and Goldberg, J.B. (2002). *Design and Analysis of Lean Production Systems.* John Wiley & Sons, New York, NY.

13. Azadivar, F. (1992). A Tutorial on Simulation Optimization. *Proceedings of the 1992 Winter Simulation Conference, J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, eds.*, 198-204.

14. Azadivar, F. (1999). Simulation Optimization Methodologies. *Proceedings of the 1999 Winter Simulation Conference, P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, eds.*, 93-100.

15. Azadivar, F., Shu, J. and Ahmad, M. (1996). Simulation Optimization in Strategic Location of Semi-Finished Products in a Pull-Type Production System. *Proceedings of the 1996 Winter Simulation Conference, J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain, eds.*, 1123-1128.

16. Barton, R.R. (1998). Simulation Metamodels. *Proceedings of the 1998 Winter Simulation Conference, D.J. Medeiros, E.F. Watson, J.S. Carson, and M.S. Manivannan, eds.*, 167-174.

17. Battiti, R. (1992). First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method. *Neural Computation*, 4: 141-166.

18. Bebis, G. and Georgiopoulos, M. (1994). Feed-forward Neural Networks: Why Network Size is So Important. *IEEE Potentials*, October/November, 27-31.

19. Bertsekas, D.P. (1999). *Nonlinear Programming, 2nd ed.* Athena Scientific, Belmont, Massachusetts.

20. Bielunska-Perlikowski, K. (1997). *Planning, Control and Management of Multicellular Manufacturing Systems by Production Authorization Cards (PAC) System.* Ph.D. Thesis, Technical University of Nova Scotia.

21. Bielunska-Perlikowski, K. and Gunn, E.A. (2002). Structure of a Simulation Model for a Manufacturing System Coordinated by Production Authorization Cards. *Journal of Design and Manufacturing Automation*, 1(4):231-245.

22. Bollon, J.-M., Di Mascolo, M., and Frein, Y. (2004). Unified Framework for Describing and Comparing the Dynamics of Pull Control Policies. *Annals of Operations Research*, 125: 21-45.

23. Bonvik, A.M., Couch, C.E., and Gershwin, S.B. (1997). A Comparison of Production-line Control Mechanisms. *International Journal of Production Research*, 35(3): 789-804.

24. Bos, S. and Chng, E.-S. (1996). Using Weight Decay to Optimize the Generalization Ability of a Perceptron. *IEEE International Conference on Neural Networks, Volume 1*, 241-6.

25. Brennan, R.W. and Rogers, P. (1995). Stochastic Optimization Applied to a Manufacturing System Operation Problem. *Proceedings of the 1995 Winter Simulation Conference, C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman, eds.*, 857-864.

26. Brown, R.G. (1967). *Decision Rules for Inventory Management*. Holt, Rinehart and Winston, New York.

27. Bulgak, A.A. and Sanders, J.L. (1988). Integrating a Modified Simulated Annealing Algorithm with the Simulation of a Manufacturing System to Optimize Buffer Sizes in Automatic Assembly Systems. *Proceedings of the 1988 Winter Simulation Conference, M. Abrams, P.Haigh, and J.Comfort, eds.*, 684-690.

28. Buzacott, J.A. (1972). The Effect of Station Breakdowns and Random Processing Times on the Capacity of Flow Lines with In-Process Storage. *AIIE Transactions*, 4(4):308-312.

29. Buzacott, J.A. and Shanthikumar, J.G. (1992). A General Approach for Coordinating Production in Multiple-Cell Manufacturing Systems. *Production and Operations Management*, 1(1):34-52.

30. Buzacott, J.A. and Shanthikumar, J.G. (1993). *Stochastic Models of Manufacturing Systems*. Prentice-Hall, Englewood Cliffs, New Jersey.

31. Carson, Y. and Maria, A. (1997). Simulation Optimization: Methods and Applications. *Proceedings of the 1997 Winter Simulation Conference, S. Andradottir, K.J. Healy, D.H. Withers and B.L. Nelson, eds.*, 118-126.

32. Chaveesuk, R. and Smith, A.E. (2003). Economic Valuation of Capital Projects Using Neural Network Metamodels. *The Engineering Economist*, 48(1): 1-30.

33. Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, 2: 303-314.

34. Dallery, Y., David, R. and Xie, X.-L. (1989). Approximate Analysis of Transfer Lines with Unreliable Machines and Finite Buffers. *IEEE Transactions on Automatic Control*, 34(9):943-953.

35. Dallery, Y. and Liberopoulos, G. (2000). Extended Kanban Control System: Combining Kanban and Base Stock. *IIE Transactions*, 32: 369-386.

36. Deleersnyder, J.-L., Hodgson, T.J., King, R.E., O'Grady, P.J. and Savva, A. (1992). Integrating Kanban Type Pull Systems and MRP Type Push Systems: Insights from a Markovian Model. *IIE Transactions*, 24(3): 43-56.

37. Demuth, H. and Beale, M. (2001). *Neural Network Toolbox User's Guide*, The MathWorks, Natick, MA.

38. Dennis, J.E. and Schnabel, R.B. (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall Inc., Englewood Cliffs, New Jersey.

39. Evans, G.W., Stuckman, B. and Mollaghasemi, M. (1991). Multicriteria Optimization of Simulation Models. *Proceedings of the 1991 Winter Simulation Conference, B.L. Nelson, W.D. Kelton, G.M. Clark, eds.*, 894-900.

40. Friedman, L.W. and Pressman, I. (1988). The Metamodel in Simulation Analysis: Can It Be Trusted? *Journal of the Operational Research Society*, 39(10): 939-948.

41. Frein, Y., DiMascolo, M., and Dallery, Y. (1995). On the Design of Generalized Kanban Control Systems. *International Journal of Operations & Production Management*, 15(9): 158-184.

42. Fu, M.C. (1994). A Tutorial Review of Techniques for Simulation Optimization. *Proceedings of the 1994 Winter Simulation Conference, J.D. Tew, M.S. Manivannan, D.A. Sadowski, A.F. Seila, eds.*, 149-156.

43. Gallant, S.I. (1993). *Neural Network Learning and Expert Systems*. The MIT Press, Cambridge, Massachusetts.

44. Gaury, E.G.A., Kleijnen, J.P.C. and Pierreval, H. (2001). A Methodology to Customize Pull Control Systems. *Journal of the Operational Research Society*, 52(7):789-799.

45. Gershwin, G. (1987). An Efficient Decomposition Method for the Approximate Evaluation of Tandem Queues with Finite Storage Space and Blocking. *Operations Research*, 35(2): 291-305.

46. Gershwin, S.B. (1994). *Manufacturing Systems Engineering.* PTR Prentice Hall, Englewood Cliffs, N. J.

47. Glover, F., Kelly, J.P. and Laguna, M. (1999). New Advances for Wedding Optimization and Simulation. *Proceedings of the 1999 Winter Simulation Conference, P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, eds.*, 255-260.

48. Glynn, P.W. (1986). Optimization of Stochastic Systems. *Proceedings of the 1986 Winter Simulation Conference, J. Wilson, J. Henriksen, S. Roberts, eds.*, 52-59.

49. Gonda Neddermeijer, H.; van Oortmarssen, G.J.; Piersma, N.; Dekker, R. (2000). A Framework for Response Surface Methodology for Simulation Optimization. *Proceedings of the 2000 Winter Simulation Conference, J.A. Joines, R.R. Barton, K. Kang, P.A. Fishwick, eds.*, Vol. 1, 129-136.

50. Gong, W.-B., Ho, Y.- C., Zhai, W. (1999). Stochastic Comparison Algorithm for Discrete Optimization with Estimation. *SIAM Journal on Optimization*, 10(2): 384-404.

51. Grosfeld-Nir, A., Magazine, M., and Vanberkel, A. (2000). Push and Pull Strategies for Controlling Multistage Production Systems. *International Journal of Production Research*, 38(11): 2361-2375.

52. Haddock, J. and Mittenthal, J. (1992). Simulation Optimization using Simulated Annealing. *Computers & Industrial Engineering*, 22(4): 387-395.

53. Hagan, M.T., Demuth, H.B., and Beale, M. (1996). *Neural Network Design.* PWS Publishing, Boston.

54. Hagan, M.T. and Menhaj, M.B. (1994). Training Feedforward Networks with the Marquardt Algorithm. *IEEE Transactions on Neural Networks*, 5(6): 989-993.

55. Hassibi, B. and Stork, D.G. (1992). Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. *Advances in Neural Information Processing, Vol. 5*, S. Hanson, J. Cowan, L. Giles (eds.), Morgan-Kaufmann, 164-171.

56. Haykin, S. (1994). *Neural Networks.* Macmillan, Englewood Cliffs, NJ.

57. Helton, J.C. and Davis, F.J. (2003). Latin Hypercube Sampling and the Propagation of Uncertainty in Analyses of Complex Systems. *Reliability Engineering & System Safety*, 81(1): 23-69.

58. Hooke, R. and Jeeves, T.A. (1961). "Direct Search" Solution of Numerical and Statistical Problems. *Journal of the ACM*, 8: 212-229.

59. Hopp, W.J. and Spearman, M.L. (2001). *Factory Physics, 2nd edition*. McGraw-Hill, Boston.

60. Huang, C.-C. and Kusiak, A. (1998). Manufacturing Control with a Push-Pull Approach. *International Journal of Production Research*, 36(1): 251-275.

61. Hurrion, R.D. (1997). An Example of Simulation Optimisation Using a Neural Network Metamodel: Finding the Optimum Number of Kanbans in a Manufacturing System. *Journal of the Operational Research Society*, 48: 1105-1112.

62. Hurrion, R.D. and Birgil, S. (1999). A Comparison of Factorial and Random Experimental Design Methods for the Development of Regression and Neural Network Simulation Metamodels. *Journal of the Operational Research Society*, 50: 1018-1033.

63. Jain, A.K., Mao, J. and Mohiuddin, K.M. (1996). Artificial Neural Networks: A Tutorial. *Computer*, 29(3): 31-44.

64. Johansson, E.M., Dowla, F.U., and Goodman, D.M. (1992). Backpropagation Learning for Multilayer Feed-Forward Neural Networks Using the Conjugate Gradient Method. *International Journal of Neural Systems*, 2(4): 291-301.

65. Kilmer, R.A., and Smith, A.E. (1993). Using Artificial Neural Networks to Approximate a Discrete Event Stochastic Simulation Model. In *Intelligent Engineering Systems Through Artificial Neural Networks, Vol. 3, ASME Press, New York, NY*, 631-636.

66. Kilmer, R.A., Smith, A.E. and Shuman, L.J. (1999). Computing Confidence Intervals for Stochastic Simulation Using Neural Network Metamodels. *Computers & Industrial Engineering*, 36(2): 391-407.

67. Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983). Optimization by Simulated Annealing. *Science*, 220(4598): 671-680.

68. Kleijnen, J.P.C. (1979). Regression Metamodels for Generalizing Simulation Results. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9 (2): 93-96.

69. Kleijnen, J.P.C., Sanchez, S.M., Lucas, T.W., and Cioppa, T.M. (2005). A User's Guide to the Brave New World of Designing Simulation Experiments. *INFORMS Journal on Computing*, 17(3): 263-289.

70. Kleijnen, J.P.C. and Sargent, R.G. (2000). A Methodology for Fitting and Validating Metamodels in Simulation. *European Journal of Operational Research*, 120(1): 14-29.

71. Kleinrock, L. (1975). *Queueing Systems Volume 1: Theory.* Wiley, New York.

72. Kleywegt, A.J. and Shapiro, A. (2001). Stochastic Optimization. In *Handbook of Industrial Engineering: Technology and Operations Management, 3rd edition*, G. Salvendy (ed.), John Wiley & Sons, New York, NY, pp. 2625-2649.

73. Kwok, T.-Y. and Yeung, D.-Y. (1997). Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems. *IEEE Transactions on Neural Networks*, 8(3): 630-645.

74. Law, A.M. and Kelton, W.D. (2000). *Simulation Modeling and Analysis, 3rd edition.* McGraw-Hill.

75. LeCun, Y., Bottou, L. Orr, G.B., and Muller, K.-R. (1998). Efficient BackProp. In *Neural Networks: Tricks of the Trade, Orr. G. and Muller, K.-R. (Eds). Lecture Notes in Computer Science 1524,* Springer-Verlag, 9-17.

76. L'Ecuyer, P. (1991). An Overview of Derivative Estimation. *Proceedings of the 1991 Winter Simulation Conference, B.L. Nelson, W.D. Kelton, G.M. Clark, eds.,* 207-217.

77. Lenard, J.D. and Roy, B. (1995). Multi-item Inventory Control: A Multicriteria View. *European Journal of Operational Research*, 87: 685-692.

78. Levenberg, K. (1944). A Method for the Solution of Certain Non-linear Problems in Least Squares," *Quarterly of Applied Mathematics.* 2:164-168.

79. Liberopoulos, G. and Dallery, Y. (2000). A Unified Framework for Pull Control Mechanisms in Multi-stage Manufacturing Systems. *Annals of Operations Research*, 93: 325-355.

80. Lippman, R.P. (1988). Neural Nets for Computing. *1988 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1-6.

81. Lutz, C.M, David, K.R., and Sun, M. (1998). Determining Buffer Location and Size in Production Lines using Tabu Search. *European Journal of Operational Research*, 406: 301-316.

82. Madu, C.N. (1990). Simulation in Manufacturing: A Regression Metamodel Approach. *Computers & Industrial Engineering*, 18(3): 381-389.

83. Markham, I.S., Mathieu, R.G. and Wray, B.A. (2000). Kanban Setting through Artificial Intelligence: A Comparative Study of Artificial Neural Networks and Decision Trees. *Integrated Manufacturing Systems*, 11(4): 239-246.

84. Marquardt, D.W. (1963). An Algorithm for Least-squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2): 431-441.

85. Martin, A.D., Chang, T.-E., Yih, Y., and Kincaid, R.K. (1998). Using Tabu Search to Determine the Number of Kanbans and Lotsizes in a Generic Kanban System. *Annals of Operations Research*, 78: 201-217.

86. Masson, E. and Wang, Y.-J. (1990). Introduction to Computation and Learning in Artificial Neural Networks. *European Journal of Operational Research*, 47: 1-28.

87. Matsuoka, K. and Yi, J. (1991). Backpropagation Based on the Logarithmic Error Function and Elimination of Local Minima. *International Joint Conference on Neural Networks*, Volume 2, 1117-1122.

88. Mollaghasemi, M. and Evans, G.W. (1994). Multicriteria Design of Manufacturing Systems Through Simulation Optimization. *IEEE Transactions on Systems, Man and Cybernetics*, 24(9):1407-1411.

89. Mollaghasemi, M., LeCroy, K., and Georgiopoulos, M. (1998). Application of Neural Networks and Simulation Modeling in Manufacturing Design. *Interfaces*, 25(5): 100-114.

90. Moller, M.F. (1993). A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning. *Neural Networks*, 6: 525-533.

91. Nasereddin, M. and Mollaghasemi, M. (1999). The Development of a Methodology for the Use of Neural Netowrks and Simulation Modeling in System Design. *Proceedings of the 1999 Winter Simulation Conference, P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, eds.*, 537-542.

92. Paris, J.-L., and Pierreval, H. (2001). A Distributed Evolutionary Simulation Optimization Approach for the Configuration of Multiproduct Kanban Systems. *International Journal of Computer Integrated Manufacturing*, 14(5): 421-430.

93. Patterson, D.A. (1996). *Artificial Neural Networks: Theory and Applications.* Simon & Schuster (Asia), Singapore.

94. Pierreval, H. and Paris, J.-L. (2000). Distributed Evolutionary Algorithms for Simulation Optimization. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans*, 30(1):15-24.

95. Pierreval, H. and Tautou, L. (1997). Using Evolutionary Algorithms and Simulation for the Optimization of Manufacturing Systems. *IIE Transactions*, 29:181-189.

96. Robinson, S.M. (1996). Analysis of Sample-Path Optimization. *Mathematics of Operations Research,* 21(3): 513-528.

97. Rohwer, R. (1991). Time Trials on Second-Order and Variable-Learning-Rate Algorithms. *Advances in Neural Information Processing, Vol. 3, R. Lippmann, J. Moody, D. Touretzky (eds.), Morgan-Kaufmann*, 977-983.

98. Rosa-Hatko, W. and Gunn, E.A. (1997) Queues with Switchover – A Review and Critique. *Annals of Operations Research*, 69(1): 299-322.

99. Ross, S.M. (1997). *Introduction to Probability Models.* Academic Press, San Diego, California.

100. Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning Internal Representations by Error Propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations, D.E. Rumelhart, J.L. McClelland (eds.), MIT Press*, 318-362.

101. Rumelhart, D.E., Widrow, B. and Lehr, M.A. (1994). The Basic Ideas in Neural Networks. *Communications of the ACM*, 37(3): 87-92.

102. Saarinen, S., Bramley, R. and Cybenko, G. (1993). Ill-Conditioning in Neural Network Training Problems. *SIAM Journal on Scientific Computing*, 14(3): 693-714.

103. Saito, K. and Nakano, R. (2000). Second-Order Learning Algorithm with Squared Penalty Term. *Neural Computation*, 12: 709-729.

104. Sammons, S.M. and Cochran, J.K. (1996). The Use of Simulation in the Optimization of a Cellular Manufacturing System. *Proceedings of the 1996 Winter Simulation Conference, J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain, eds.*, 1129-1134.

105. Sargent, R.G. (1991). Research Issues in Metamodeling. *Proceedings of the 1991 Winter Simulation Conference, B.L. Nelson, W.D. Kelton, G.M. Clark, eds.*, 888-893.

106. Savsar, M. and Choueiki, M.H. (2000). A Neural Network Procedure for Kanban Allocation in JIT Production Control Systems. *International Journal of Production Research*, 38(14): 3247-3265.

107. Sengupta, S., Sharief, F. and Dutta, S.P. (1999). Determination of the Optimal Number of Kanbans and Kanban Allocation in a FMS: A Simulation-based Study. *Production Planning & Control*, 10(5):439-447.

108. Silver, E.A., Pyke, D.F., and Peterson, R. (1998). *Inventory Management and Production Planning and Scheduling*, 3rd edition. John Wiley & Sons, New York.

109. Smith, A.E. and Mason, A.K. (1997). Cost Estimation Predictive Modeling: Regression versus Neural Network. *The Engineering Economist*, 42(2):137-161.

110. Smith, M. (1993). *Neural Networks for Statistical Modeling.* Van Nostrand Reinhold, New York.

111. Starr, M.K. and Miller, D.W. (1962). *Inventory Control: Theory and Practice.* Prentice-Hall, Englewood Cliffs, New Jersey.

112. Stinchcombe, M. and White, H. (1990). Approximating and Learning Unknown Mappings Using Multilayer Feedforward Networks with Bounded Weights. *International Joint Conference on Neural Networks*, Volume 3: 7-16.

113. Swisher, J.R., and Jacobson, S.H. (1999). A Survey of Ranking, Selection, and Multiple Comparison Procedures for Discrete-Event Simulation. *Proceedings of the 1999 Winter Simulation Conference, P.A. Farrington, H.B. Nembhard, D.T. Sturrock, and G.W. Evans, eds.*, 492-501.

114. Tekin, E. and Sabuncuoglu, I. (2004). Simulation Optimization: A Comprehensive Review on Theory and Applications. *IIE Transactions*, 36(11): 1067-1081.

115. Tompkins, G. and Azadivar, F. (1995). Genetic Algorithms in Optimizing Simulated Systems. *Proceedings of the 1995 Winter Simulation Conference, C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman, eds.*, 757-762.

116. Trafalis, T. (1999). Primal-Dual Optimization Methods in Neural Networks and Support Vector Machines Training. *Advanced Course on Artificial Intelligence (ACAI99), Machine Learning & Applications*, Chania, Greece.

117. Twomey, J.M. and Smith, A.E. (1998). Bias and Variance of Validation Methods for Function Approximation Neural Networks Under Conditions of Sparse Data. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews,* 28(3): 417-430.

118. Wysk, R.A., Yang, N.S. and Joshi, S. (1991). Detection of Deadlocks in Flexible Manufacturing Cells. *IEEE Transactions on Robotics and Automation*, 7(6): 853-859.

119. Yakowitz, S., L'Ecuyer, P., and Vazquez-Abad, F. (2000). Global Stochastic Optimization with Low-Dispersion Point Sets. *Operations Research*, 48(6): 939-950.

120. Yan, D. and Mukai, H. (1992). Stochastic Discrete Optimization. *SIAM Journal on Control and Optimization*, 30(3): 594-612.

121. Yu, B. and Popplewell, K. (1994). Metamodels in Manufacturing: A Review. *International Journal of Production Research*, 32(4): 787-796.

Appendix A

PAC Simulation Model (PACSIM)

The PAC simulation model, PACSIM, is written in FORTRAN 77, and includes some of the routines and functions from the SIMLIB routines of Law & Kelton (2000). The starting point for the development of PACSIM was the simulation model developed by Bielunska-Perlikowski (1997). The major components of PACSIM are explained in this Appendix. The programs can be found in Appendix I, and can be compiled using any FORTRAN compiler (we used Absoft ProFortran 7.5).

A.1 Main Program Flow

The main calling program in the simulation model is *pacsim.f*. It calls the subroutine PAC to execute one simulation run. Design points obtained from the input file may be run multiple times, as specified in the input file *today.txt*. Each time a simulation is executed, output is written to either *pac.out*, or the measurement files, *m_*.txt*, or both, based again on specifications in the file *today.txt*. An overview of *pacsim.f* is shown in Figure A.1.

A.1.1 Input Files

Several input files are required to run the simulations. There are three declaration files:

- *pac.dcl*: Declares common variables used throughout the simulation subroutines
- *system.dcl*: includes parameters set for use throughout the model, based on the manufacturing system to be simulated
- *param.dcl*: includes number of lists and statistical variables to be used in SIMLIB. These parameters are used to declare array sizes in *pac.dcl*.

The description of the manufacturing system to be simulated is found in the input file *pact.in*. A new worksheet (Figure A.2) was created in Microsoft Excel to allow for easier

entry of the data required, and the generation (through use of a macro) of the file *pact.in* in the correct format (Figure A.3). This worksheet also calculates the parameter values for the *system.dcl* and *param.dcl* files (Figure A.4).

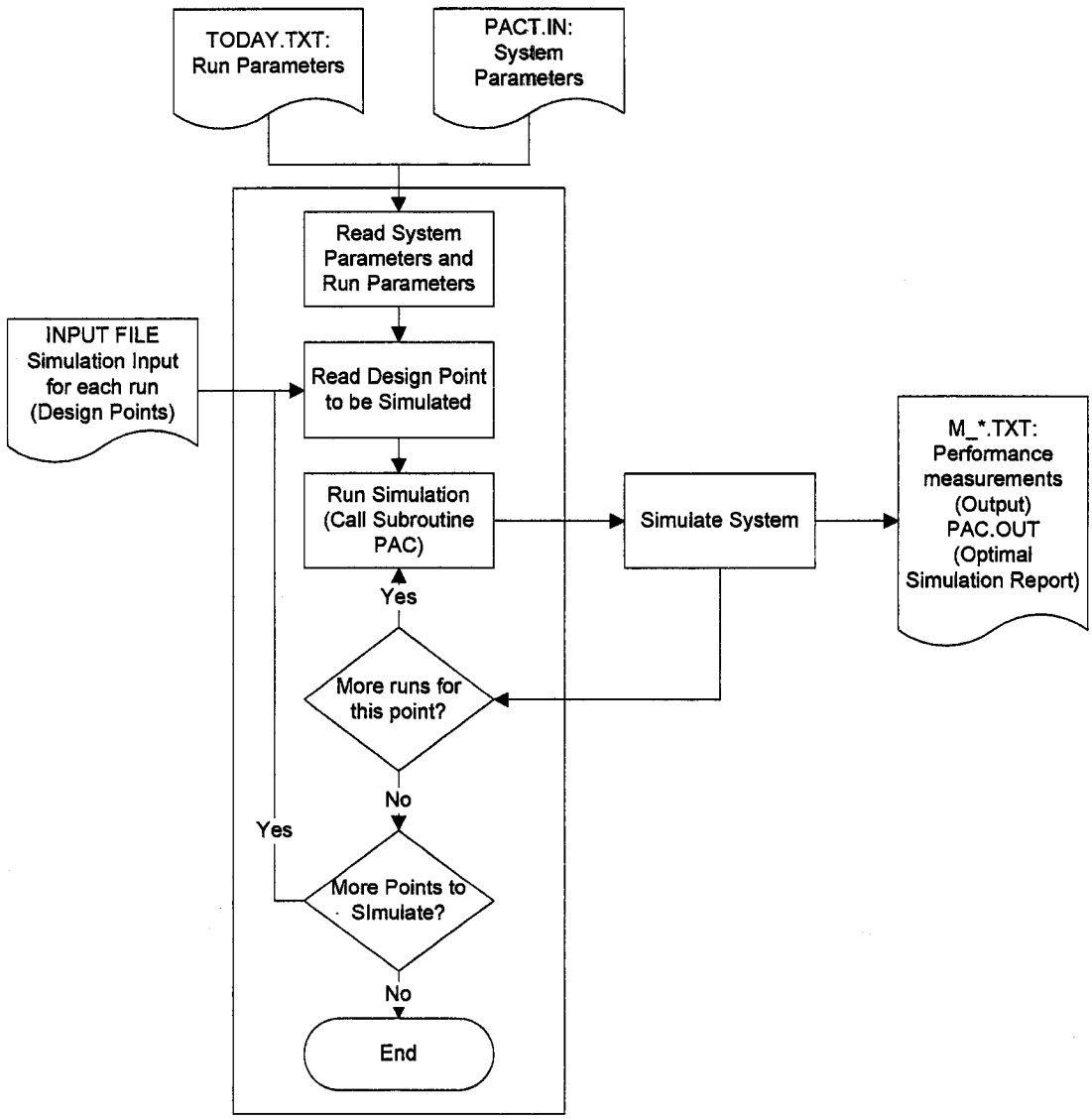

Figure A.1: Overview of Program Flow (*pacsim.f*)

Number of Cells	Number of Final Type Goods	Number of Final/Assy Types	Number of Sub-Assy	Number of Raw Matl Parts	b/w demand arrivals	Length of Sim run in days	Warmup period in days	Code for Control Type
NUNITS	FTYPES	MTYPES	ATYPES	RTYPES	MARRVT	LENGTH	WARMUP	CODE
2	1	0	1	1	60.0	250.0	50.0	3

Number of machines in each cell:

Cell NO:	1	2
Machines:	1	1

For each manufactured product:

Product No.	z	k	r	τ	Mean Service Time	Cell at which it is produced	Count of SubAssy's needed	Names of SubAssy's 1st,2nd,...	Qty of SubAssy 1st, 2nd,...
1	2	1	1	0	36.0	2	1	2	1
2	0	1	1	0	45.0	1	1	3	1

For each final or final/assy part, the probability that an arriving demand is for product I

Prod NO:	1
Probability:	1.0

Enter the number of transportation time records

2

Transportation Entries

Enter Product Number	Cell it is going to	Transport Time	Cost Factor $/item/day
2	2	0.0	4.0
3	1	0.0	4.0

Product No.	Delay Cost $/item/day	
1	10.0	For each Final or Final/Assy Item, Enter the cost of delay in meeting demand ($/item/day)

Product No.	Holding Cost $/item/day	Setup Time per product	
1	4.0	0.0	For all manufactured products, input the cost of holding in product store,
2	4.0	0.0	and setup time at downstream cell

Figure A.2: Worksheet for Simulation System Data (Model 0)

```
2,1,0,1,1,60.0,250.0,50.0,3
1,1
2,1,1,0,36.0,2,1,2,1
0,1,1,0,45.0,1,1,3,1
1.0
2
2,2,0.0,4.0
3,1,0.0,4.0
10.0
4.0,0.0
4.0,0.0
```

Figure A.3: Contents of the *pact.in* File Produced by the Worksheet (Model 0)

For Param.DCL:		
MA	5	
MR	40000	
ML	23	$(m * 6) + [2 * (f + fa)] + n + [m * (n + r)] + 1$
ST	10	$m + 2n + r + 3 * (f + fa)$
MT	33	ML + ST
For System.DCL:		
MU	2	m
MP	2	n
MG	3	n + r
MS	1	

Figure A.4: Parameter Values for Declaration Files Provided by Worksheet (Model 0)

If only one scenario is to be simulated, the design point (the *z, k, r* and τ values for each cell/store) can be specified in the *pact.in* file. However, if multiple design points are to be simulated, a file containing these points must be provided to the model. The name of the file containing the design points is specified in the file *today.txt*, which is a new file created to easily change run variables between runs of the simulation model (Figure A.5). This file also specifies the name of the parameter file (usually *pact.in*), whether reports are required, the number of simulation runs per design point, and other variables.

```
pact.in
Input_ModelA.txt
2     #RPT:        Reports: 1: report, 2: data files, 3: both
0     #DBUG:       Debug statements:  0: turn off, 1: turn on
2     #SCENS:      Scenarios:  1: Just one, use pact.in, 2: read from input file
1     #RUNS:       Number of runs (replications) per design point
0     #TRANSIENT:  Transient Check?  1: Write to transient.txt  0: Otherwise
2     #DIST:       Type of distribution:  1-Exponential, 2-Weibull, alpha=2
0     #BATCHING:   Input includes r values (batching)?  1:Yes 0:No
```

Figure A.5: Contents of today.txt File

A.2 Main Simulation Subroutine (PAC)

The main simulation subroutine, PAC, is contained in PAC.F. This subroutine is called for each simulation run for each design point to be simulated. The PACOPT.F program passes the PAC values (design point) to this subroutine, and PAC executes one simulation run. The simulation is a discrete-event simulation model. The PAC subroutine handles the system clock, and calls the appropriate subroutine to process each of the 7 system events may occur (as discussed in Section A.2.2).

A.2.1 Entities in the Simulation

There are five groups of entities which can be created in the simulation model.

- **Product/Part:** The initial inventory parameters, z_i, determine the number of parts or products created at each store at the beginning of the simulation. Once a part (or group of parts) has undergone processing at a cell, each component entity is destroyed and a new entity (the completed part or product) is created. If the entity is a finished product, it leaves the system when requisitioned by a customer.
- **Orders**: Order entities are generated by a customer order (demand) or the arrival of a PA card at a downstream cell. If no batch production is required, and a process tag is available, then the order entity is immediately destroyed when the PA card is created. If no process tags are available, or if batches of orders are required to form PA cards, then the order entities remain at the store until these requirements are satisfied.

- **Requisitions**: Requisitions for products or parts are created after a delay, τ, from receipt of the order for the product. They remain at the appropriate store until the requested product arrives, at which point they are destroyed.
- **Process tags:** These entities are created at each store at the beginning of the simulation, according the parameters set for the simulation experiment. Although in reality process tags would never be destroyed, in the simulation they are destroyed when removed from the queue at the cell, and then created again when the PA card originally generated returns to the store with the processed part and no orders are waiting for matching.
- **Production Authorization (PA) cards**: PA card entities are created when an order entity is matched with an available process tag entity. They are sent to the appropriate production cell where they remain until the part authorized to be produced is completed and sent to the requesting store, at which time they are destroyed.

A.2.2 Simulation Events

There are seven simulation events (Table A.1) which may occur in the simulation model. The event graph for the simulation is shown in Figure A.6.

Table A.1: Event List for the PAC Simulation

Event	Description
1	Arrival of a customer order or part order from a cell
2	Arrival of a requisition at a store
3	Arrival of a Production Authorization card at a cell
4	Part and Process tag arrival at a store
5	Completion of processing of a part at a cell
6	Arrival of work in process inventory at a cell
7	End of the simulation

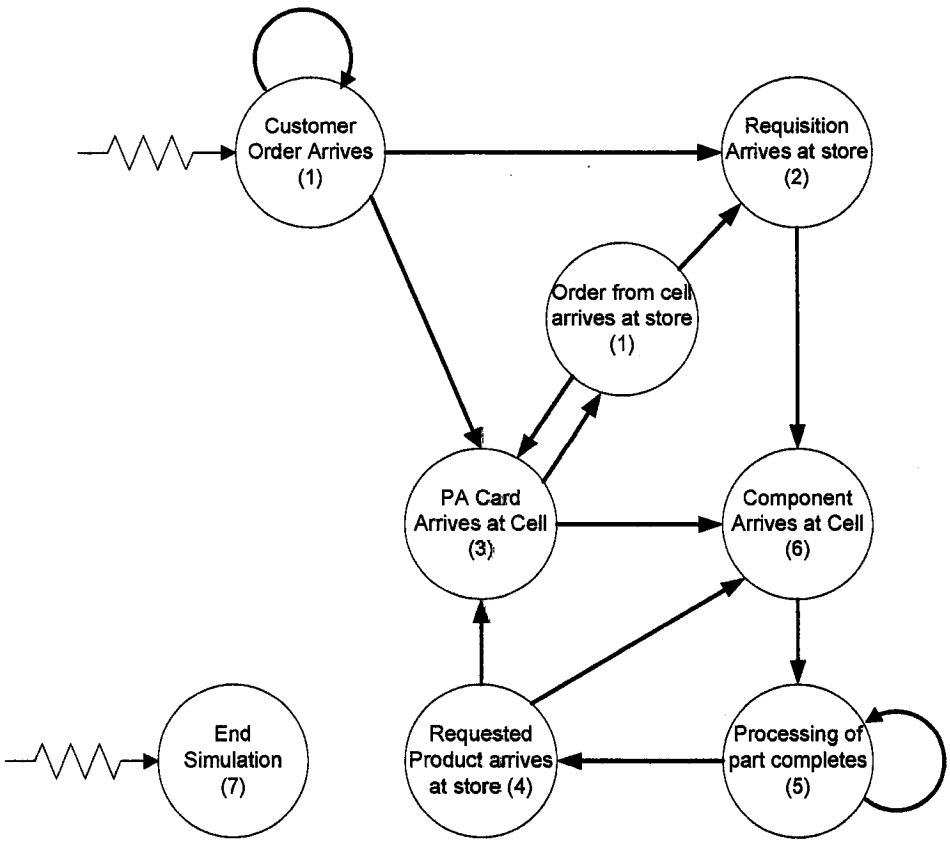

Figure A.6: Event Graph for the PAC Simulation

Each event is handled by a separate subroutine called by the PAC subroutine. An overview of the program flow for the PAC subroutine, including the main subroutines called, is shown in Figure A.7.

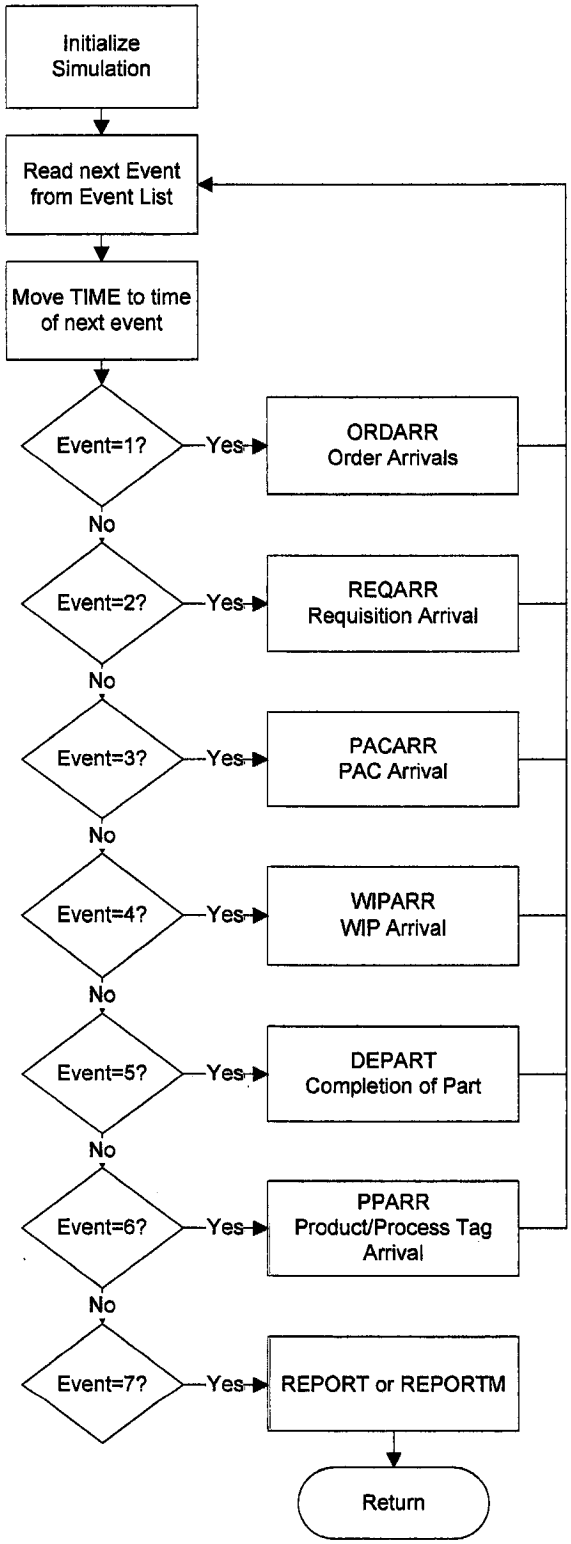

Figure A.7: Overview of Program Flow for pac.f

A.2.3 Description of Subroutines

A.2.3.1 Arrival of an Order (ORDARR)

Orders can either be customer orders (NEW=1) or orders from cells (NEW=2). If the order is from a customer, ORDARR immediately generates the next customer order and adds this to the event list. Next, a requisition arrival event is added to the event list to request the ordered part or product, with the time of arrival equal to τ minutes from the current time. Finally, if a process tag is available for the part ordered, a production authorization card arrival event for the appropriate cell is added to the event list.

A.2.3.2 Arrival of a Requisition (REQARR)

When a requisition arrival event occurs, the REQARR subroutine checks to see if there is product available at the store; if there is, the number of products at the store is reduced by one. If the requisition was from a customer, then the product leaves the system; otherwise, REQARR creates a work-in-process arrival event at the requesting cell, with the time of arrival equal to the current time plus the travel delay time, if any.

A.2.3.3 Arrival of a Process Authorization Card (PACARR)

When a PA card arrival event occurs, PACARR immediately generates order arrival events for all of the required components at the upstream cells, unless the required component is a raw material, in which case the routine creates a WIP arrival event (type 6) at the cell to occur after the travel delay time for the material, if any.

A.2.3.4 Arrival of a completed part and process tag (PPARR)

The subroutine PPARR handles the arrival of a completed part and process tag at a store. Upon arrival, the subroutine checks to see if there are any orders waiting for a matching process tag; if there are, it then generates a PA card arrival at the appropriate cell. It then checks to see if there are any requisitions waiting for the part. If there is a waiting requisition from a cell, a work-in-process arrival event is generated for the requesting cell, to occur after any travel delay. If a requisition is waiting from a customer, it is immediately filled and the product leaves the system. If no requisition is waiting, the part is added to the inventory at the store.

A.2.3.5 Departure of a Completed Part a Cell (DEPART)

When the end of processing occurs at a cell, the completed part must then be sent to the requesting location; therefore, a type 4 event is added to the event list, to process the arrival of the finished part and the process tag to the requesting store. The original PA card which authorized the production of this part is destroyed. The routine then checks to see if there are more jobs waiting at the cell for processing, and if so, begins processing and schedules the future departure of the next product; if not, the status of the machine is changed to idle.

A.2.3.6 Arrival of a part at the requesting cell (WIPARR)

The arrival of a part requested by a cell is processed by the WIPARR subroutine. When a component is requested, the product for which it has been requested is recorded. Therefore, upon the arrival of the component, the routine checks to see that all other required components have also arrived; if not, the component waits in the cell queue. If all required parts have arrived, and if the cell is not currently busy, production starts and the routine creates the departure event for the part currently being processed.

A.2.3.7 End of the Simulation (REPORT or REPORTM)

At the end of the simulation, either the subroutine REPORT or REPORTM is called. REPORT calls the statistical routines in SIMLIBG to do final the final statistical calculations, and produces a simulation report (*pac.out*). REPORTM only calculates the values which are written to the measurement files *m_*.txt*, which are described in Table A.2. These files were added to the model to provide the training data for the neural networks.

Table A.2: Input and Output Files and Programs for the Simulation

File	Contents
m_fillrt.txt	Average fill rate for each finished product for each simulation run
m_ct.txt	Average cycle time per finished product for each simulation run
m_fgi.txt	Average inventory for each finished product for each simulation run
m_cdelay.txt	Average time in the queue for customer orders for each product for each run
m_wipNQ.txt	Average work-in-process inventory for all intermediate parts for each run
Transient.txt	Optional data file for transient analysis
pac.out	Simulation Report

A.2.4 Cycle Time

For manufacturing systems where assembly stations are present, this means that the model will have to determine the "oldest" component in order to determine the overall cycle time, and track this value for each product produced. We define cycle time as the time from the start of production on a product to the time it arrives in finished goods inventory. As such, it is a means to approximate the work in process inventory in the system, as cycle time and WIP (when throughput remains constant) are directly proportional. In systems with no assembly stations, this single measure can be used instead of the individual WIP amounts at each station.

A.3 SIMLIB subroutines of Law & Kelton (2000)

The following routines from the SIMLIB set of routines (Law & Kelton, 2000) are used in the PAC simulation model. These routines are used by the PAC model to maintain the event list, queues and other lists, and to track variables needed for statistical calculations.

A.3.1 File and Statistical Variable Subroutines

- INITLK: Initializes the file lists
- FILE (OPTION, LIST): Adds a record to the appropriate list. OPTION determines how the item is added (ie. at the beginning, at the end, etc.)
- REMOVE (OPTION, LIST): Removes a record from the list. OPTION determines if the record removed is the first or the last record in the list.
- TIMING: This routine removes the first (next) record from the event list. The system time is then updated to the occurrence time of this event.
- SAMPST (VALUE, VARIBL): A statistical routine which maintains a running total of some value and the number of observations of this value. It also maintains the maximum and minimum observation of this variable to this point. When called with a negative value of VARIBL, the routine reports the results of these observations, as well as the average value of the observations of the variable. For example, the routine is called every time a part is completed on a

machine, so that the average time to complete the part can be calculated at the end of the simulation.

- TIMEST (VALUE, VARIBL): This routine calculates the time-averaged value of a variable. The routine is called every time the value of a variable is changed. For example, the average number of parts waiting in a store is calculated using this routine, and the routine is called every time a part leaves or arrives at the store. When called with a negative value of VARIBL, it does the final calculations for VARIBL and reports the results.
- FILEST (LIST): This routine is called at the end of the simulation in order to call TIMEST with VARIBL set to the negative value of LIST.

A.3.2 Random number generation

Function RANFN(ISTRM) is the random number generator provided with SIMLIB. It produces a pseudo-random variable with distribution U(0,1). ISTRM is the seed value.

The following subroutines call RANFN(ISTRM) to obtain a random variable which is uniformly distributed between zero and one, and then use that value and the inverse of the appropriate distribution function, to generate a random variate.

- EXPON(RMEAN,ISTRM): Generates an random variate from an exponential distribution with mean RMEAN.
- WEIBULL(RBETA,ISTRM): Generates a random variate from a Weibull distribution with β = RBETA and $\alpha = 2$.
- IRANDI (NVALUE, PROB, ISTRM): This routine first obtains a U(0,1) random variate from stream ISTRM. It then converts this value into an integer between 1 and NVALUE in accordance with the cumulative distribution function PROB.
- UNFRM(A,B,ISTRM): Generates a random variate from a uniform distribution with a = A and b = B.

A.4 Queues and Lists

The master array of SIMLIB maintains queues and lists. Queues contain entities, and the lists manage additional information for statistical purposes, as well as the master event list for the simulation.

At each cell/store, there are six individual queues maintained (Table A.3).

Table A.3: Queues Maintained by SIMLIB

Queue	Entity Stored	Description
ORD	Orders	If an order for a product arrives at a store, and no product tags are available, the order is added to this queue. Also, if batching is involved, this queue holds orders which may have already been matched to a process tag, but are waiting for the required number of orders to arrive in order to form the batch of PA cards.
REQ	Requisitions	This queue holds requisition entities awaiting the arrival of the requested part/product. The number of requisitions waiting in this queue represents the backlog.
PROD	Products/Parts	All stock at a store which has not yet been requested by a customer or downstream cell.
PROC	Process Tags	Process tags available at a store for matching with orders
WIP	Products/Parts	Parts requested by the cell for processing, but which are awaiting processing at the cell. The reason entities may be held here is that the requesting cell may be already busy, or for an assembly operation, other required components have not yet arrived.
PAC	PA cards	Active production authorization cards at the cell (for product awaiting processing or in process at the cell)

In addition to the queues, the program maintains several lists which track events and entities. The main list is the event list, which maintains all upcoming events by event type. The remaining lists are not queues, but rather track the existence of entities in the corresponding queues in Table A.3. This was done for statistical purposes. These lists are shown in Table A.4.

Table A.4: Lists Maintained by SIMLIB

List	Description
CUST	When a requisition for a finished product arrives, and the demand cannot be immediately met, a requisition entity is created and placed in the REQ queue at the store, and a corresponding entry is made in this list. When the REQ entity is destroyed, this list entry is also removed.
CORD	When an order for a finished product arrives, and the demand cannot be immediately met, an order entity is created and placed in the ORD queue at the store, and a corresponding entry is made in this list. When the ORD entity is destroyed, this list entry is also removed.
PPRD	If a production cell produces more than one part/product type, then the associated product store (PROD queue) will contain different product types. There is a separate PPRD list for each product type produced by system, and each time a part/product entity is added to a PROD queue, a list entry is made in the corresponding PPRD list.
PWIP	The WIP queue at a processing cell contains all part/product entities awaiting processing at the cell, and therefore may contain different product types; there is a separate PWIP list for each product type at each cell, and each time a part entity is added to the WIP queue, a list entry is made in the corresponding PWIP list for that part
EVENT	The event list

Each record maintained in each of the queues or lists contain up to five values. For the entities in a queue, these are the attributes of the entity. The attributes maintained for each entry in the lists are described below and shown in Table A.5.

Table A.5: Entity Attributes

	Attribute 1	Attribute 2	Attribute 3	Attribute 4	Attribute 5
ORD	Arrival time	Product type	Matched with process tag		
REQ	Arrival time	Product type	Address		
PROD	Arrival time	Product type			Start Time
PROC	Arrival time	Product type			
WIP	Arrival time	Product type			Start Time
PAC	Arrival time	Product type			
CUST	Arrival time				
CORD	Arrival time				
PPRD	Arrival time	Product type			Start Time
PWIP	Arrival time	Product type			Start Time
EVENT	Event time	Event type	Product type	Address	Other

"Address" refers to the cell to which the requisition entity is to be delivered (this is relevant in the case where multiple cells produce the same product). "Matched with

process tag" refers to whether the order is waiting in the queue because no product tags are available (value 0) or whether the order has been matched to a process tag but is waiting until sufficient orders arrive to form a batch (value 1). "Start time" is used for products to maintain the total time to produce the product, and therefore this value is the start time of the first component produced for this product. In the case of an assembled product, this value is the start time of the oldest component.

Table A.6 shows the lists and queues maintained in the simulation, and the list address within the master array.

Table A.6: List Addresses

List	Number of Lists	Addressed by
ORD*	NUNITS each	1+(STORE-1)*6
REQ*		2+(STORE-1)*6
PROD*		3+(STORE-1)*6
PROC*		4+(STORE-1)*6
WIP*		5+(CELL-1)*6
PAC*		6+(CELL-1)*6
CUST	FTYPES + MTYPES	NUNITS*6+PRODTYPE
PPRD	NTYPES	NUNITS*6+FTYPES+MTYPES+PRODTYPE
PWIP	NUNITS* (NTYPES+RTYPES)	NUNITS*6+FTYPES+MTYPES+NTYPES+ (CELL-1)*(NTYPES+RTYPES) +PRODTYPE
CORD	FTYPES+MTYPES	NUNITS*6+FTYPES+MTYPES+NTYPES+ NUNITS*(NTYPES+RTYPES) +PRODTYPE
EVENT	1	NUNITS*6+2*(FTYPES+MTYPES)+NTYPES+ NUNITS*(NTYPES+RTYPES) +1

A.5 System States

The only state which needs to be maintained is the state of each machine at each cell. A variable, NBUSY(I) maintains the number of busy machines at each cell.

A.6 Statistics

The average number of entities in a queue, and the minimum and maximum values observed are maintained for each queue in Table A.3. As well, the time-averaged number of entries in the lists of Table A.4, and the minimum and maximum observed

values, are also maintained. In addition to these statistics, there are other statistical variables tracked by the simulation model. These are described in Table A.7.

Table A.7: Other Statistical Variables

Description of Variable	Number of Variables	Subroutine used
Delay in Cell (on machines)	NUNITS	SAMPST
Delay in WIP Queue, by product (first ones for finished goods not used)	NTYPES+RTYPES	SAMPST
Customer Delay from time of requisition. arrival	FTYPES+MTYPES	SAMPST
Customer Delay, from time of order arrival	FTYPES+MTYPES	SAMPST
Delay in product stores	NTYPES	SAMPST
Cycle time	FTYPES+MTYPES	SAMPST
Number of Machines busy in a Cell	NUNITS	TIMEST

The subroutine SSTATE was added to the model to track some of the system measures to determine whether, for the design point provided, the system could reach steady state. This routine is called by several of the event subroutines each time a customer order arrives, a customer order is filled, or a finished product arrives at a final inventory point. The results from these calculations are written to a separate data file by either REPORT or REPORTM. A sample output report, *pac.out*, is shown below for a single simulation of Model A.

Sample pac.out full report:

```
**************************************************************************
*   SIMULATION OF MULTIPLE-CELL SYSTEM COORDINATED BY PA CARDS (PAC)     *
*   Report created on: 03-Aug-05                                         *
**************************************************************************

 INPUT DATA
 ----------
 Number of cells/stores                3
 Number of machines in each cell       1    1    1
 Number of "final"          products        1
```

```
Number of "final/assembly" products        0
Number of "assembly"       products        2
Number of "raw materials"                  1
Distr.funct. of "final" product types 1.00
Mean interarr.time of all "final" prod.   60.00 min.
Length of the simulation                 250.0  24-hours days
Length of "warming up"                    50.0  24-hours days

Parameter settings for the coordination scheme:

 Product  Cell/Store  Mean service time  z-value  k-value  r-value   t-value
  type                   (in min.)
 =======  ==========  ================  =======  =======  =======  ========
   1          3           40.00            2        4        1         0
   2          2           36.00            2        3        1         0
   3          1           42.00            2        1        1         0

Product type   No. of subassemblies   Subassembly name   Units of subassembly
============   ====================   ================   ====================
      1                 1                   2                    1
      2                 1                   3                    1
      3                 1                   4                    1

Product    Cell    Time to transport a unit of product    WIP Cost
 type              from storage to cell (in min.)      ($/day/item)
=======    ====    ===================================  ==============
   2         3                  5.00                       4.00
   3         2                  5.00                       4.00
   4         1                  0.00                       2.00

Product           Customer Service Cost
 type            Delay Cost ($/item/day)
=========================================
   1                          10.00

Product       PROD Cost         Setup
 type       ($/day/item)     (in min.)
=======================================
   1             4.00           0.00
   2             4.00           0.00
   3             4.00           0.00


 SIMULATION RESULTS
 ------------------

 SERVICE LEVEL
 -------------
(calculated from time of arrival of requisition tag)

 Product           Probability an arriving demand
  type           from a customer is met immediately

   1                          0.576

 Product        Delay in meeting a customer demand     Number of
  type             Average     Max          Min          units

   1              34.521     482.031       0.000      4695.000
```

Overall average product total delay to customer = 34.521

SERVICE LEVEL

(calculated from time of arrival of order tag)

Product type	Delay in meeting a customer demand Average	Max	Min	Number of units
1	34.521	482.031	0.000	4695.000

Overall average product total delay to customer = 34.521

Product type	Cell	Delay (waiting time) in Product Store Average	Max	Min
1	3	52.861	602.953	0.000
2	2	57.929	653.594	0.000
3	1	56.123	648.484	0.000

Product type	Delay (waiting time) in WIP -queue Average	Max	Min
2	28.009	208.562	0.000
3	17.793	139.562	0.000
4	0.000	0.000	0.000

Machines in cell	Average number in queue	Average utilization	Average delay in queue
1	0.000	0.690	0.000
2	0.290	0.597	17.793
3	0.457	0.649	28.009

Product type	Completion Time from start of production Average	Max	Min
1	288.609	1035.359	82.133

INVENTORY LEVELS

PROD-queues (per store)	Time average	Maximum	Minimum
1	0.915	2.000	0.000
2	0.944	2.000	0.000
3	0.862	2.000	0.000

Time average inventory of parts in stores : 2.721

PPRD-queues (per product)	Time average	Maximum	Minimum
1	0.862	2.000	0.000
2	0.944	2.000	0.000
3	0.915	2.000	0.000

WIP -queues (per cell)	Time average	Maximum	Minimum
1	0.000	1.000	0.000
2	0.290	2.000	0.000
3	0.457	3.000	0.000

Time average work-in-process inventory in cells: 0.747

PWIP-queues product	cell	Time average	Maximum	Minimum
2	3	0.457	3.000	0.000
3	2	0.290	2.000	0.000

Total WIP waiting in queues/stores by product

Product	Number in Queues or Stores
2	1.401
3	1.205
4	0.000

STATISTICAL SUMMARY DATA ON RECORDS IN QUEUES

ORD -queues (per store)	Time average	Maximum	Minimum
1	0.680	4.000	0.000
2	0.136	3.000	0.000
3	0.181	9.000	0.000

REQ -queues (per store)	Time average	Maximum	Minimum
1	0.285	3.000	0.000
2	0.333	4.000	0.000
3	0.564	11.000	0.000

PROC-queues (per store)	Time average	Maximum	Minimum
1	0.310	1.000	0.000
2	1.747	3.000	0.000
3	2.480	4.000	0.000

PAC -queues (per cell)	Time average	Maximum	Minimum
1	0.690	1.000	0.000
2	1.253	3.000	0.000
3	1.520	4.000	0.000

CUST-queues (per product)	Time average	Maximum	Minimum
1	0.564	11.000	0.000

CORD-queues (per product)	Time average	Maximum	Minimum
1	0.564	11.000	0.000

CUST. "REQ" SERV.COST [$] =	1125.53	(28.9 %)
CUST. "ORD" SERV.COST [$] =	1125.53	(28.9 %)
PROD. INVENTORY COST [$] =	2176.78	(55.8 %)
WIP. INVENTORY COST [$] =	597.26	(15.3 %)
TOTAL COST [$] =	3899.57	(100.0 %)

Number of Replications: 1
Total run time: (seconds): 0.109
(minutes): 0.002

A.7 Preprocessing (preprocess.f)

This FORTRAN 77 program produces a report to help determine the minimum values for the design point values which will allow the simulation to reach steady state.

A.7.1 Determination of Feasible Design Points

The simulation model is run multiple times for a set of design points with very low numbers of process tags and all initial inventory points set to zero. The preprocessing program is then run to report the results of these simulations. This report, *pp_REPORT.txt* (shown on the next page), contains the results of several simulation runs each point. Information reported includes:

- **Percent Diff**: Percentage difference between demand arrivals and system throughput. A large, negative value indicates that the system could not keep up with demand.

$$\text{Percent Diff} = \frac{Ord - Dep}{Ord}$$

where

Ord = Average time between arrivals of customer orders
Dep = Average time between arrivals of finished goods inventory at store

- **Reg Line**: The slope of a linear regression line fitted to the customer delay time versus time. A large value indicates that the customer delay continues to increase as time passes, once again indicating infeasibility. Although this relationship, in the case of an unstable system probably non-linear, a linear regression line fitted to these points will still have a positive slope.
- **Customer Delay Time**: The overall average of customer delay time is reported. As well, the program determines the average for all observations occurring before the half-way point of the simulation, as well as the average after this point, and reports those values. If the average of the second half is significantly larger than the first half, this once again indicates infeasibility.

These results are reported for each finished product produced by the system. The decision criteria were determined based on observation.

```
                    Preprocessing Report
                  Created on:           20-Oct-05

          Number of RUNS per Design Point:   5
          Length of each simulation run:    200 days


 Design point tested:   0   1   0   1   0   1

Product No.  1

    Time BTW Time BTW   Percent   Reg       Customer Delay Time        Cycle
Run  Departs   Orders    Diff    Line    Average  1st Half  2nd Half   Time

  1  128.157   61.304 -109.05  2.93509  115148.6   76874.0  153731.1   128.2
  2  129.241   59.794 -116.14  2.83922  110642.1   72764.0  148554.2   129.2
  3  128.533   60.135 -113.74  2.92068  114917.1   75868.0  153447.0   128.5
  4  128.572   60.135 -113.81  2.92988  114959.1   75485.3  153458.2   128.6
  5  127.748   60.049 -112.74  2.81208  109478.4   70469.5  148660.7   127.8
                     ----------------------------------------------------
          Average:   -113.10  2.88739  113029.1   74292.2  151570.2   128.4

          Percent Difference less than -0.200%
          Avg. Regression slope above  0.150
     2nd half delay average is 104.0% higher 1st half delay average

     ***Design point probably INFEASIBLE ***


 Design point tested:   5   5   5   5   5   5

Product No.  1

    Time BTW Time BTW   Percent   Reg       Customer Delay Time        Cycle
Run  Departs   Orders    Diff    Line    Average  1st Half  2nd Half   Time

  1   59.810   59.847    0.06  0.00009       5.0       6.7       3.3   683.7
  2   59.914   59.920    0.01  0.00006       3.2       4.1       2.2   685.5
  3   60.667   60.633   -0.06  0.00011       4.6       3.2       5.9   691.7
  4   57.583   57.594    0.02  0.00016       6.8       4.8       8.7   665.4
  5   59.972   59.982    0.02  0.00007       2.7       2.2       3.2   685.2
                     ----------------------------------------------------
          Average:      0.01  0.00010       4.5       4.2       4.7   682.3

              Design point feasible


      -- SUMMARY --

Feasible?         Design Point

* NO *          0  1  0  1  0  1
 Yes            5  5  5  5  5  5
```

A.8 Testing Design Points (*testreports.f*)

During analysis, it was often necessary to compare the results of one design point simulated multiple times with the results from the neural network. This program generates a test report (shown below, for only one design point) for such purposes. It also calculates confidence intervals for the simulation results.

```
                    Test Report
              with  99% Confidence Intervals
                 Created on:          20-Oct-05

        Number of RUNS per Design Point:   5
        Length of each simulation run:    200 days


 Design point tested:   5   5   5   5   5   5

Product No.  1

Run      Cust       Cycle      Fill      FG
         Delay      Time       Rate      Inv
  1       15.02    711.23    0.8500    3.141
  2       26.49    696.34    0.8190    2.993
  3       20.13    697.80    0.8370    3.023
  4       21.17    689.81    0.8300    3.066
  5       15.05    700.76    0.8460    3.065
      ------------------------------------
Avg       19.57    699.19    0.8364    3.058

LCL       14.04    690.15    0.8220    2.993
UCL       25.10    708.22    0.8508    3.122


WIP Results, by Product


Run      Prod 2   Prod 3   Prod 4
  1       4.571    3.905    1.059
  2       4.412    3.960    1.133
  3       4.448    3.895    1.125
  4       4.344    3.773    1.238
  5       4.473    3.941    1.107
      ------------------------
Avg       4.450    3.895    1.132

LCL       4.407    3.857    1.099
UCL       4.493    3.932    1.166
```

A.9 Generating the Design Points (generate_random.f)

The purpose of this program is to generate the dataset of design points to be used as input to the simulation model and for training the neural networks. The logic of the program is shown in Figure A.8.

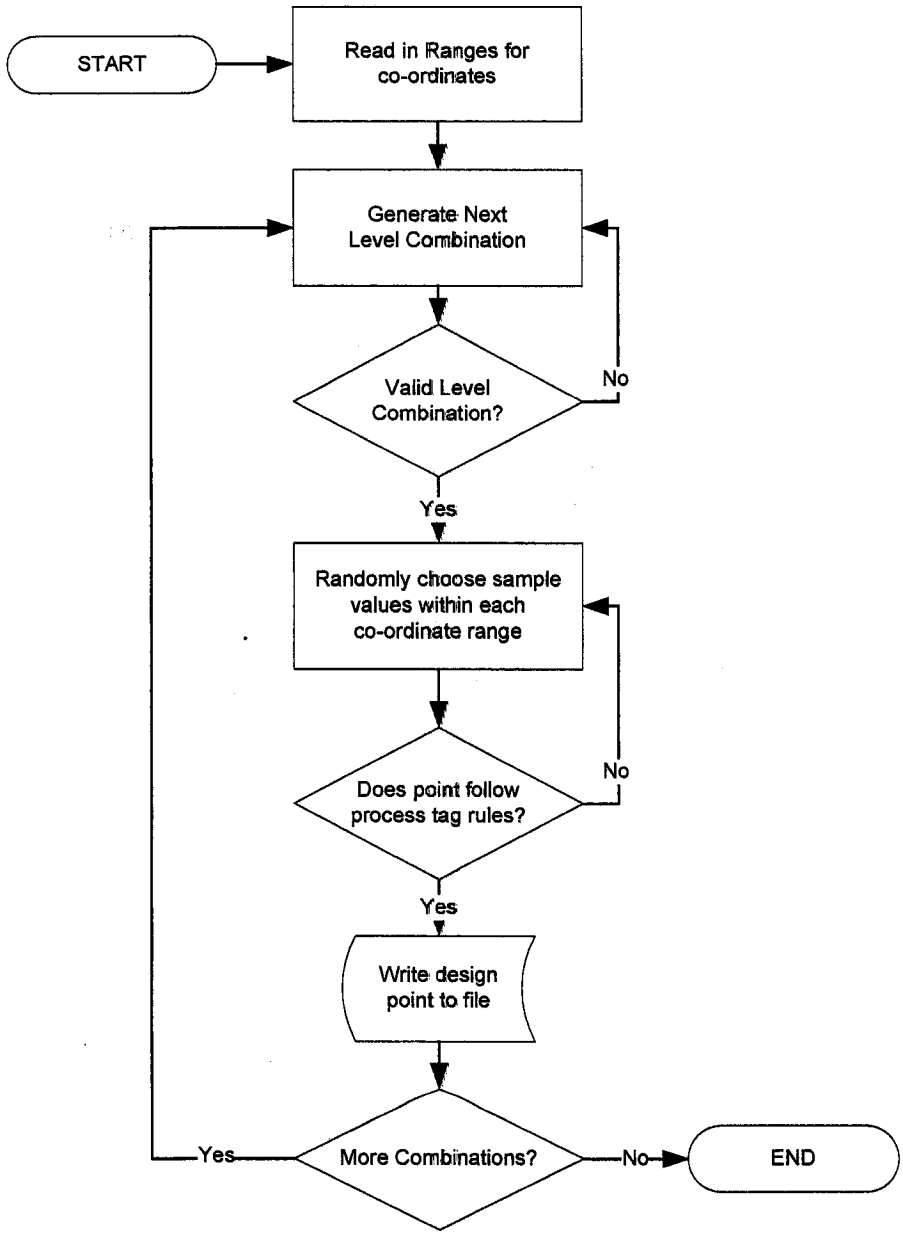

Figure A.8: Overview of Program Logic for Generation of Design Points (*generate_random.f*)

A.10 Transient Period Analysis

This program, *transient.f*, processes the file *transient.txt* which is produced by the simulation model when the switch in *today.txt* is set to 1. The file produced prepares the data in a table format for easier import into MS Excel for analysis.

A.11 Procedure for Experiments

An overview of the procedure for producing a training dataset for a model and then training neural networks for the model is shown in Figure A.9.

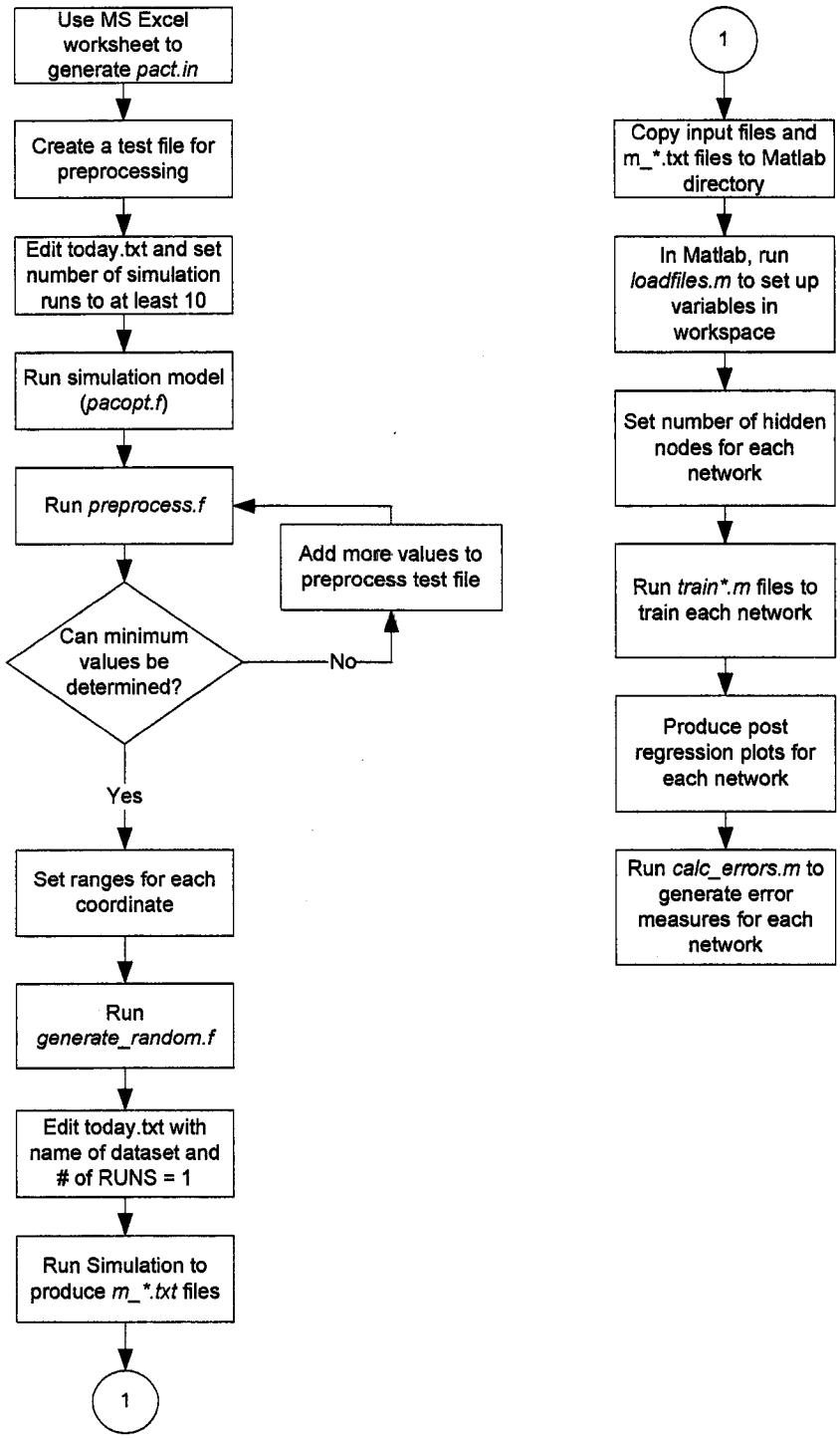

Figure A.9: Flowchart of Procedure for Generating a Training Set and Training Networks

Appendix B

Neural Network Formulas

In a training data set, there are N example pairs, X_n, t_n, with inputs $X_n = \{x_{1n}, x_{2n}, \ldots, x_{In}\}$ and the corresponding observed output, t_n.

The feedforward calculations for each example input n are:

$$u_{jn} = a_{0j} + \sum_I a_{ij} x_{in} + \alpha_j$$

$$y_{jn} = \frac{1}{1 + \exp(-u_{jn})}$$

$$v_n = b_0 + \sum_J b_j y_{jn}$$

where

a_{ij} = weight on input from input node i to hidden node j
b_{jk} = weight on output of hidden node j to output node k
α_j = symmetry breaking constants for hidden node j (optional)

The difference between the network output for the n^{th} example at the output node, v_n, and the actual or target output, t_n, represents the error. The goal of learning is to minimize the error function over all of the example pairs:

$$E = \tfrac{1}{2N} \sum_N (v_{kn} - t_{kn})^2$$
$$= \tfrac{1}{2N} \left((v_1 - t_1)^2 + \cdots + (v_n - t_n)^2 + \cdots + (v_N - t_N)^2 \right)$$

Denote the n^{th} error term of E as E_n:

$$E_n = \tfrac{1}{2}(v_n - t_n)^2 \quad n = 1, \ldots, N$$

The derivative of a sum of terms is equal to the sum of the derivatives of each term. Therefore, starting with the partial derivatives of the error function with respect to the output node weights, we have

$$\frac{\partial E}{\partial b_j} = \sum_N \frac{\partial E_n}{\partial b_j}$$

Each error term, E_n, is a function of the outputs, v_n, which are the sums of the weighted inputs to the nodes, and therefore a function of those weights. Using the chain rule to find the partial derivative of each term E_n with respect to each output node weight, b_j:

$$\frac{\partial E_n}{\partial b_j} = \frac{\partial E_n}{\partial v_n} \cdot \frac{\partial v_n}{\partial b_j}$$

For the first partial derivative of E_n with respect to one output, v_{kn},

$$\frac{\partial E_n}{\partial v_n} = \frac{\partial \left[\frac{1}{2}(v_n - t_n)^2\right]}{\partial v_n} = v_n - t_n$$

Then, the partial derivative of v_n with respect to b_j:

$$v_n = b_0 + \sum_J b_j y_{jn} = b_0 + b_1 y_{1n} + \cdots + b_j y_{jn} + \cdots + b_J y_{Jn}$$

$$\frac{\partial v_n}{\partial b_j} = \frac{\partial}{\partial b_j}\left(b_0 + b_1 y_{1n} + \cdots + b_j y_{jn} + \cdots + b_J y_{Jn}\right)$$

$$= \begin{cases} 1 & \text{if } j = 0 \\ y_{jn} & \text{otherwise} \end{cases}$$

For ease of notation, assume that:

$$y_{0n} = 1 \quad \forall\, n = 1, \ldots, N$$

$$x_{0n} = 1 \quad \forall\, n = 1, \ldots, N$$

Therefore,

$$\frac{\partial v_n}{\partial b_j} = y_{jn}$$

Bringing all of the terms together,

$$\frac{\partial E_n}{\partial b_j} = \frac{\partial E_n}{\partial v_n} \cdot \frac{\partial v_n}{\partial b_j} = (v_n - t_n) y_{jn}$$

Finding the partial derivatives of the error function with respect to the input weights is somewhat more complex. Once again using the chain rule, the partial derivative of E_n with respect to a_{ij} is

$$\frac{\partial E_n}{\partial a_{ij}} = \frac{\partial E_n}{\partial y_{jn}} \cdot \frac{\partial y_{jn}}{\partial u_{jn}} \cdot \frac{\partial u_{jn}}{\partial a_{ij}}$$

To determine the first derivative, recall that

$$E_n = \tfrac{1}{2}(v_n - t_n)^2$$

Because each term in this function is itself a function of y_{jn} then first the derivative of E_n in terms of y_{jn} must be computed.

$$\frac{\partial E_n}{\partial y_{jn}} = \frac{\partial\left(\frac{1}{2}(v_n - t_n)^2\right)}{\partial y_{jn}} = \frac{\partial\left(\frac{1}{2}(v_n - t_n)^2\right)}{\partial v_n} \cdot \frac{\partial v_n}{\partial y_{jn}} = (v_n - t_n) \cdot \frac{\partial v_n}{\partial y_{jn}}$$

$$\frac{\partial v_n}{\partial y_{jn}} = \frac{\partial\left(b_0 + \sum_J b_j y_{jn}\right)}{\partial y_{jn}} = b_j$$

$$\therefore \frac{\partial E_n}{\partial y_n} = (v_n - t_n) b_j$$

The next partial derivative is

$$\frac{\partial y_{jn}}{\partial u_{jn}} = \frac{\partial}{\partial u_{jn}}\left(\frac{1}{1+e^{-u_{jn}}}\right) = y_{jn}(1 - y_{jn})$$

Finally,

$$\frac{\partial u_{jn}}{\partial a_{ij}} = \frac{\partial}{\partial a_{ij}}(a_{0j} + a_{1j}x_{1n} + \cdots + a_{ij}x_{in} + \cdots + a_{iJ}x_{In}) = x_{in}$$

where $x_{0n} = 1 \; \forall \; n$

Substituting all of these into the original equation:

$$\frac{\partial E_n}{\partial a_{ij}} = \frac{\partial E_n}{\partial y_{jn}} \cdot \frac{\partial y_{jn}}{\partial u_{jn}} \cdot \frac{\partial u_{jn}}{\partial a_{ij}}$$

$$= (v_n - t_n) b_j y_{jn}(1 - y_{jn}) x_{in}$$

To find the derivative of the original error function, E, sum the derivatives of all the error terms E_n over N. Therefore,

$$\frac{\partial E}{\partial b_{jk}} = \frac{1}{N}\sum_N \frac{\partial E_n}{\partial v_n}\cdot\frac{\partial v_n}{\partial b_j} = \frac{1}{N}\sum_N (v_n - t_n)y_{jn}$$

$$\frac{\partial E}{\partial a_{ij}} = \frac{1}{N}\sum_N \frac{\partial E_n}{\partial y_{jn}}\cdot\frac{\partial y_{jn}}{\partial u_{jn}}\cdot\frac{\partial u_{jn}}{\partial a_{ij}}$$
$$= \frac{1}{N}\sum_N [(v_n - t_n)b_j]y_{jn}(1 - y_{jn})x_{in}$$

Second derivatives for the Error function:

The following second derivatives are required:

$$\frac{\partial^2 E}{\partial a_{ij}\partial a_{i'j'}}, \frac{\partial^2 E}{\partial b_j \partial a_{i'j'}}, \frac{\partial^2 E}{\partial a_{ij}\partial b_{j'}}, \frac{\partial^2 E}{\partial b_j \partial b_{j'}}$$

To simplify, find the second derivatives of each additive term of the first derivatives, and then sum these second derivatives over N.

First derivatives:

$$\frac{\partial E_n}{\partial a_{ij}} = ((v_n - t_n)b_j)y_{jn}(1 - y_{jn})x_{in}$$

$$\frac{\partial E_n}{\partial b_j} = (v_n - t_n)y_{jn}$$

Second derivatives:

$$\frac{\partial}{\partial b_{j'}}\left(\frac{\partial E_n}{\partial b_j}\right) = \frac{\partial}{\partial b_{j'}}((v_n - t_n)y_{jn}) = y_{jn}y_{j'n}$$

$$\frac{\partial}{\partial b_{j'}}\left(\frac{\partial E}{\partial b_j}\right) = \frac{1}{N}\sum_N y_{jn}y_{j'n}$$

$$\frac{\partial}{\partial a_{i'j'}}\left(\frac{\partial E_n}{\partial b_j}\right)=\frac{\partial}{\partial a_{i'j'}}\left((v_n-t_n)y_{jn}\right) \quad where\ j'\neq 0$$

$$=\begin{cases} y_{j'n}\left(1-y_{j'n}\right)x_{i'n}\left[b_{j'}y_{jn}+(v_n-t_n)\right] & if\ j=j' \\ y_{j'n}\left(1-y_{j'n}\right)x_{i'n}\left[b_{j'}y_{jn}\right] & otherwise \end{cases}$$

$$\frac{\partial}{\partial b_{j'}}\left(\frac{\partial E}{\partial a_{ij}}\right)=\begin{cases} \frac{1}{N}\sum_N\left[y_{jn}\left(1-y_{jn}\right)x_{in}\left(b_j y_{j'n}+(v_n-t_n)\right)\right] & if\ j=j' \\ \frac{1}{N}\sum_N\left[y_{jn}\left(1-y_{jn}\right)x_{in}\left(b_j y_{j'n}\right)\right] & otherwise \end{cases}$$

$$\frac{\partial}{\partial a_{i'j'}}\left(\frac{\partial E_n}{\partial a_{ij}}\right)=\frac{\partial}{\partial a_{i'j'}}\left(\left((v_n-t_n)\cdot b_j\right)y_{jn}\left(1-y_{jn}\right)x_{in}\right) \quad where\ j\neq 0,\ j'\neq 0$$

$$=\begin{cases} y_{j'n}\left(1-y_{j'n}\right)x_{i'n}x_{in}\left[\begin{matrix}\left(b_j b_{j'}\right)y_{jn}\left(1-y_{jn}\right)\\ +\left(b_j(v_n-t_n)\right)\left(1-2y_{j'n}\right)\end{matrix}\right] & j=j' \\ y_{j'n}\left(1-y_{j'n}\right)x_{i'n}x_{in}\left[\left(b_j b_{j'}\right)y_{jn}\left(1-y_{jn}\right)\right] & otherwise \end{cases}$$

$$\frac{\partial}{\partial a_{i'j'}}\left(\frac{\partial E}{\partial a_{ij}}\right)=\begin{cases} \frac{1}{N}\sum_N\left[y_{j'n}\left(1-y_{j'n}\right)x_{i'n}x_{in}\left[\begin{matrix}\left(b_j b_{j'}\right)y_{jn}\left(1-y_{jn}\right)\\ +\left(b_j(v_n-t_n)\right)\left(1-2y_{j'n}\right)\end{matrix}\right]\right] & j=j' \\ \frac{1}{N}\sum_N\left[y_{j'n}\left(1-y_{j'n}\right)x_{i'n}x_{in}\left[\left(b_j b_{j'}\right)y_{jn}\left(1-y_{jn}\right)\right]\right] & otherwise \end{cases}$$

Partial derivatives of the neural network function with respect to the input variables:

$$\frac{\partial v}{\partial x_i}=\sum_J\frac{\partial v}{\partial y_j}\cdot\frac{\partial y_j}{\partial u_j}\cdot\frac{\partial u_j}{\partial x_i}=\sum_J b_j y_j\left(1-y_j\right)a_{ij}$$

Appendix C

Levenberg-Marquardt Approach to Network Training

The following is a description of the Levenberg-Marquardt method (Levenberg, 1944, Marquardt, 1963) for minimizing a function of least squares.

The error function to be minimized is the sum of squares function, E:

$$F(\mathbf{w}) = E = \tfrac{1}{2N}\sum_{N}(v_n - t_n)^2$$

And the derivatives of this function are

$$\frac{\partial E}{\partial a_{ij}} = \tfrac{1}{N}\sum_{N}\left[\sum_{K}(v_n - t_n)b_j\right]y_{jn}(1-y_{jn})x_{in} \qquad \frac{\partial E}{\partial b_j} = \sum_{N}(v_n - t_n)y_{jn}$$

To simplify the notation, if we let $\mathbf{w}$ be the weight vector, $\mathbf{w} = \{a_{01}, \ldots, a_{IJ}, b_0, \ldots, b_J\}^T$, where C is the number of terms in the weight vector, then

$$e_n(\mathbf{w}) = e_n = v_n - t_n$$
$$\mathbf{e} = [e_1(\mathbf{w}) \quad e_2(\mathbf{w}) \quad \ldots \quad e_N(\mathbf{w})]^T$$

Then

$$F(\mathbf{w}) = \tfrac{1}{2}\sum_{m=1}^{N} e_n^2 = \tfrac{1}{2}\mathbf{e}^T\mathbf{e}$$

then the j^{th} element of the gradient of F would be:

$$[\nabla F(w)]_c = \frac{\partial}{\partial w_c}\left(\tfrac{1}{2}\sum_{n=1}^{N} e_n^2\right) = \sum_{n=1}^{N} e_n(w)\cdot\frac{\partial e_n(w)}{\partial w_c} = \mathbf{J}_j^T\mathbf{e}$$

where $\mathbf{J}^T{}_c$ is the c^{th} row of the transposed Jacobian matrix, $\mathbf{J}$:

$$\mathbf{J}=\begin{bmatrix} \frac{\partial e_1}{\partial w_1} & \frac{\partial e_1}{\partial w_2} & \cdots & \frac{\partial e_1}{\partial w_C} \\ \frac{\partial e_2}{\partial w_1} & \frac{\partial e_2}{\partial w_2} & \cdots & \frac{\partial e_2}{\partial w_C} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_N}{\partial w_1} & \frac{\partial e_N}{\partial w_2} & \cdots & \frac{\partial e_N}{\partial w_C} \end{bmatrix} \qquad J(i,j)=\frac{\partial e_i}{\partial w_j}$$

Therefore, the gradient would be

$$\nabla \mathbf{F}(w)=\mathbf{J}^T\mathbf{e}$$

To find the Hessian matrix, the $(k,j)^{th}$ element of the matrix would be:

$$\begin{aligned}\left[\nabla^2\mathbf{F}(w)\right]_{k,j} &= \frac{\partial^2 F(w)}{\partial w_k \partial w_j}=\frac{\partial}{\partial w_k}\left[\sum_{n=1}^{N} e_n(w)\cdot\frac{\partial e_n(w)}{\partial w_j}\right] \\ &= \sum_{n=1}^{N}\left[\frac{\partial e_n(w)}{\partial w_k}\cdot\frac{\partial e_n(w)}{\partial w_j}+e_n(w)\cdot\frac{\partial^2 e_n(w)}{\partial w_k \partial w_j}\right] \\ &= \sum_{n=1}^{N}\left[\frac{\partial e_n(w)}{\partial w_k}\cdot\frac{\partial e_n(w)}{\partial w_j}\right]+\sum_{n=1}^{N}\left[e_n(w)\cdot\frac{\partial^2 e_n(w)}{\partial w_k \partial w_j}\right]\end{aligned}$$

Therefore, the Hessian matrix can be written as:

$$\nabla^2\mathbf{F}(w)=\mathbf{J}^T\mathbf{J}+\mathbf{S}$$

where $$\mathbf{S}=\sum_{n=1}^{N} e_n(w)\cdot\frac{\partial^2 e_n(w)}{\partial w_k \partial w_j}$$

If we assume **S** to be small, then we can approximate the Hessian as

$$\nabla^2\mathbf{F}(w)\approx\mathbf{J}^T\mathbf{J}$$

Substituting these approximations in to the original Newton method we obtain the Gauss-Newton Method (s = current stage):

Steepest Descent:

$$\mathbf{w}_{s+1}=\mathbf{w}_s-\alpha_s\nabla\mathbf{F}(\mathbf{w}_s)$$

Newton:

$$\mathbf{w}_{s+1}=\mathbf{w}_s-\left(\nabla^2\mathbf{F}(\mathbf{w}_s)\right)^{-1}\nabla\mathbf{F}(\mathbf{w}_s)$$

Gauss-Newton

$$\mathbf{w}_{s+1} = \mathbf{w}_s - \left(\mathbf{J}^T(\mathbf{w}_s)\mathbf{J}(\mathbf{w}_s)\right)^{-1}\mathbf{J}^T(\mathbf{w}_s)\mathbf{e}(\mathbf{w}_s)$$

With this method, $\mathbf{J}^T\mathbf{J}$ may not be positive definite, and therefore not invertible. This can be overcome by using the following modification to the Hessian approximation (Hagan et al., 1996, p. 12-21):

$$\mathbf{G} = \mathbf{J}^T\mathbf{J} + \mu\mathbf{I}$$

If $\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ are the eigenvalues of the matrix $\mathbf{J}^T\mathbf{J}$, then $\{\lambda_1+\mu, \lambda_2+\mu \ldots, \lambda_n+\mu\}$ are the eigenvalues of $\mathbf{G}$. Therefore, $\mathbf{G}$ can be made positive definite by increasing μ until $(\lambda_i + \mu)>0$ for all i (convexification).

Therefore, the new update method becomes

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}^T(\mathbf{w}_k)\mathbf{J}(\mathbf{w}_k) + \mu_k\mathbf{I}\right)^{-1}\mathbf{J}^T(\mathbf{w}_k)\mathbf{e}(\mathbf{w}_k)$$

This is known as the Levenberg-Marquardt method.

The value of μ_k can be changed at each iteration. As μ_k approaches zero, the method approaches the Gauss-Newton method. If μ_k is large, the technique approaches that of the steepest descent method, with small step size α.

The Levenberg-Marquardt backpropagation algorithm (LMBP) (Hagan et al., 1996) is as follows:

1. Do a forward pass through the network with current weights. Calculate the squared error.

2. Compute the Jacobian matrix.

3. Find next weight vector:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}^T(\mathbf{w}_k)\mathbf{J}(\mathbf{w}_k) + \mu_k\mathbf{I}\right)^{-1}\mathbf{J}^T(\mathbf{w}_k)\mathbf{e}(\mathbf{w}_k)$$

4. Recompute the squared error with $\mathbf{w}_{k+1}$. If the new sum of squared errors is less than the previous value, then divide μ by ν (a predefined constant) and to go Step 1. Otherwise, multiply μ by ν and go back to step 3.

Appendix D

Results for Model 0

D.1 Transient Analysis

Forty replications of the simulation at the point $z_1 = 0$, $k_1 = 2$, $z_2 = 0$, $k_2 = 1$ were conducted. If Y_{ij} represents the observation for customer wait time for the i^{th} observation from the j^{th} replication of the simulation, then the average for the i^{th} observation across all J observations is calculated, in this case, as

$$\bar{Y}_i = \frac{1}{40}\left(\sum_{j=1}^{40} Y_{ij}\right)$$

These values were used to plot a moving average (w =100) of average customer delay time across replications (Figure D.1). The x-axis on this graph has been converted to days (approximately) by using the information that customer orders arrive, on average, at a rate of 1 order per hour, and there are therefore, on average, 24 observations per day. From this graph, it can be seen that the simulation begins to level off around 20 days, however 50 days was chosen as the warmup period to ensure steady state results.

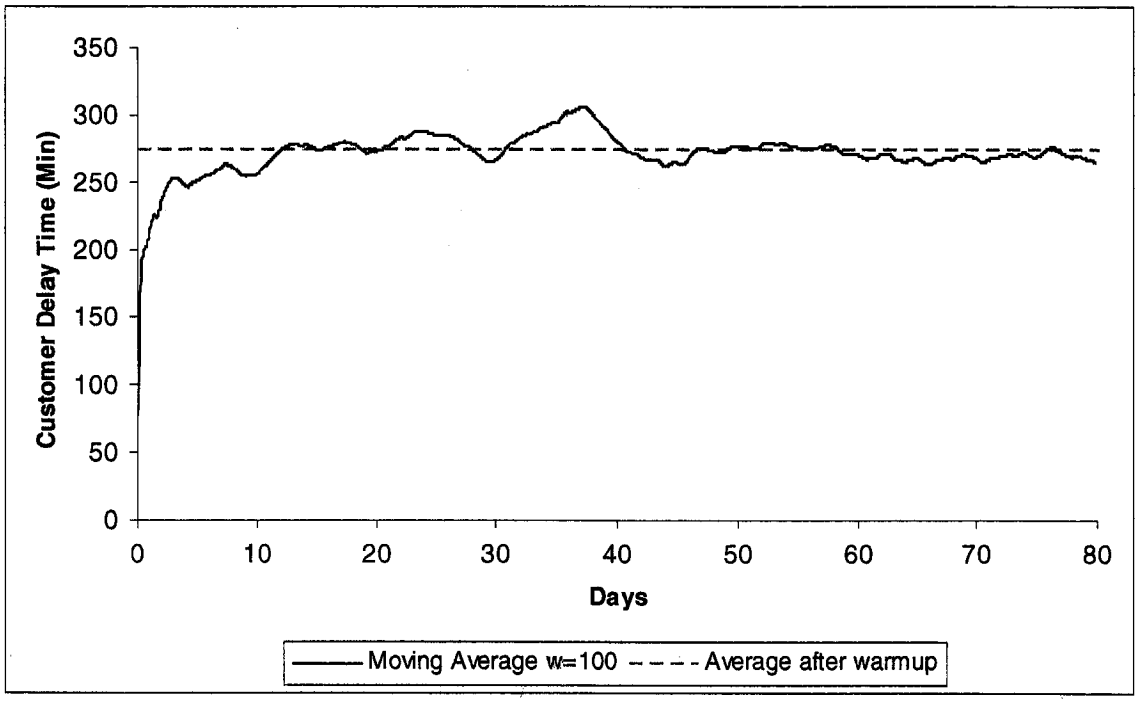

Figure D.1: Welch's Graphical Procedure for Customer Delay Time, Model 0

D.2 Simulation Model Verification

Table D.1: Verification Results, Model 0

Replication	X_j	$(\overline{X} - X_j)^2$	Replication	X_j	$(\overline{X} - X_j)^2$
1	243.38	701.35	21	258.58	127.43
2	260.40	89.55	22	278.36	72.26
3	265.14	22.30	23	270.34	0.22
4	292.90	530.81	24	276.19	40.04
5	259.61	105.14	25	254.42	238.36
6	276.57	44.95	26	273.84	15.78
7	281.65	138.86	27	299.84	898.76
8	277.84	63.68	28	259.06	116.79
9	273.89	16.20	29	252.23	311.01
10	271.24	1.90	30	294.58	611.13
11	256.48	179.16	31	282.87	169.13
12	269.57	0.09	32	279.47	92.33
13	302.61	1072.20	33	271.96	4.41
14	271.70	3.39	34	275.55	32.38
15	267.02	8.08	35	254.44	237.82
16	273.41	12.54	36	282.93	170.65
17	267.57	5.27	37	247.07	519.36
18	262.88	48.79	38	260.10	95.33
19	253.29	274.76	39	272.50	6.94
20	269.14	0.52	40	253.92	254.34

$$\sum_{j=1}^{40} X_j = 10794.55$$

$$\sum_{j=1}^{40} (\overline{X} - X_j)^2 = 7334.00$$

$$\overline{X} = \frac{\sum_{j=1}^{40} X_j}{40} = \frac{10794.55}{40} = 269.86$$

$$S^2(40) = \frac{\sum_{j=1}^{40} (\overline{X} - X_j)^2}{39} = \frac{7334.00}{39} = 188.05$$

$$CI = \overline{X} \pm t_{39,0.975}\sqrt{\frac{S^2(40)}{40}} = 269.86 \pm 2.021\sqrt{\frac{188.05}{40}} = 269.86 \pm 4.38$$

Table D.2: Network Training Parameters, Model 0

Network	Hidden Nodes	Training Epochs
Customer Delay	12	300
Cycle Time	15	200
Fill Rate	15	250
FGI	12	150

D.3 Network Weights

Network 1: Customer Delay Time

Table D.3: Network 1 Weights, Model 0

Hidden Layer							Output Layer	
		Input Node i						Output
		0	1	2	3	4		
Hidden Node j	1	-0.1365	-0.59292	-0.07768	-0.39032	-0.21801	0	0.98299
	2	0.53888	-0.29162	-0.28026	-0.29337	-0.07048	1	0.67515
	3	0.34897	0.17181	0.27223	0.15394	0.27231	2	0.51218
	4	0.29458	0.02017	0.22607	0.16813	0.2577	3	0.17278
	5	-0.13076	1.5143	0.40121	-0.48334	0.02632	4	0.24368
	6	-0.16404	-0.24623	-0.12269	0.30081	0.0811	5	0.75087
	7	-0.53093	0.04939	-0.56618	0.56472	0.00186	6	0.33263
	8	-0.32739	-0.07152	-0.27533	0.27558	-0.2746	7	0.47862
	9	0.35519	-0.15865	-0.05944	-0.11027	-0.04693	8	0.39842
	10	6.9168	2.2384	0.65408	3.2332	0.02643	9	0.44672
	11	-2.962	-2.1371	0.13743	-0.08502	0.08331	10	-4.4404
	12	0.82506	-0.04588	0.52261	0.39442	0.26234	11	1.6993
							12	0.56744

Network 2: Cycle Time

Table D.4: Network 2 Weights, Model 0

Hidden Layer							Output Layer	
		Input Node i						Output
		0	1	2	3	4		
Hidden Node j	1	0.82103	-0.10167	0.11828	-0.92958	-0.06921	0	0.09054
	2	0.86626	-0.09441	0.14117	-0.80613	-0.14253	1	-0.60791
	3	0.60113	-0.27041	0.79396	-0.21885	0.28959	2	-1.0552
	4	0.63895	-0.0447	0.07318	-0.35513	-0.27645	3	0.5835
	5	-0.18311	0.14882	-0.29408	-0.2952	0.3106	4	-0.97123
	6	0.05087	-0.26607	-0.46907	0.33522	-0.35901	5	0.95796
	7	0.07934	-0.56956	-0.04898	0.41717	0.15876	6	-0.6623
	8	0.28027	-0.35946	-0.75094	-0.56635	0.52092	7	-0.75034
	9	0.81658	0.02289	0.09881	-1.2288	0.13899	8	0.72854
	10	-0.20393	-0.10539	0.00996	1.4802	0.02945	9	-1.2064
	11	-0.15582	0.11981	-0.10303	0.01437	1.4456	10	1.7651
	12	0.37096	-0.05947	0.30077	-0.18868	-0.99463	11	-0.74104
	13	1.9797	-0.06475	0.06037	-0.13711	1.1902	12	0.34963
	14	-0.37299	-0.39419	-0.02982	-0.29523	0.25831	13	1.5173
	15	-0.63581	0.12523	-0.56657	0.07291	0.83812	14	0.36063
							15	-0.49528

Network 3: Fill Rate

Table D.5: Network 3 Weights, Model 0

Hidden Layer							Output Layer	
		Input Node i						Output
		0	1	2	3	4		
Hidden Node j	1	0.4854	0.03175	-0.06536	-0.33353	-0.02725	0	-0.58915
	2	1.8103	3.3479	-0.01854	-0.48287	-0.12449	1	-0.52691
	3	0.21468	-0.59601	-0.00395	0.2087	-0.16958	2	-2.9196
	4	-0.10991	0.36711	-0.21622	1.1102	0.12364	3	0.55102
	5	0.17843	0.0709	0.11594	-0.16224	-0.56126	4	-0.97371
	6	0.19624	0.2763	0.06531	-0.28801	-0.1299	5	-0.57537
	7	1.0152	0.72966	-0.30114	0.29042	0.0005	6	0.03763
	8	0.06486	0.09812	-0.29209	-0.5905	0.29534	7	0.8996
	9	-0.14127	0.2322	0.50737	-0.22274	-0.22743	8	-0.88045
	10	1.5267	1.3347	0.03893	0.14914	-0.00064	9	-0.60503
	11	2.235	2.2867	0.26361	0.7	-0.11554	10	0.94358
	12	-2.7903	-2.8528	0.00188	1.4103	0.01196	11	1.2828
	13	-0.45315	0.04658	0.00273	0.63034	-0.17917	12	-3.0405
	14	1.8095	1.5294	-0.04992	0.07658	-0.19392	13	-0.11343
	15	3.215	4.0022	-0.07959	0.94053	0.04429	14	1.0757
							15	2.3179

Network 4: Finished Goods Inventory

Table D.6: Network 4 Weights, Model 0

Hidden Layer							Output Layer	
		Input Node i						Output
		0	1	2	3	4		
Hidden Node j	1	-0.56516	0.2351	-0.06444	-0.59302	-0.1311	0	-1.3546
	2	-0.40913	0.10955	0.39025	-0.01522	-0.29184	1	-0.66416
	3	-0.15053	0.69011	-0.03186	0.73578	0.00908	2	0.19918
	4	-0.69193	1.7834	-0.01774	0.08044	0.09759	3	1.6095
	5	-0.68575	0.93533	0.07355	0.32885	-0.34907	4	1.5248
	6	0.16876	0.46054	0.37695	-0.05375	0.02622	5	0.60102
	7	0.26301	-0.89117	0.00856	-0.6366	-0.20754	6	-0.89597
	8	-0.51783	0.45065	0.00753	0.49655	-0.00179	7	-0.53943
	9	-0.44353	-0.20286	-0.06842	1.488	0.01657	8	0.88602
	10	-0.10013	-0.3244	0.13323	0.28984	0.09664	9	-2.3804
	11	0.65546	-0.12925	-0.03472	0.68761	-0.03273	10	0.76529
	12	0.53731	-0.05668	0.30803	0.24963	-0.30669	11	1.2537
							12	0.13275

D.4 Post Regression Plots

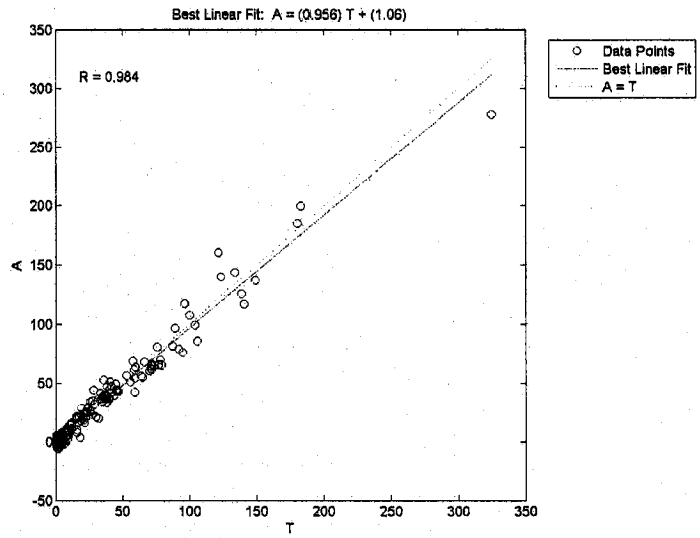

Figure D.2: Customer Delay Time (Network 1) Post Regression Plot, Model 0

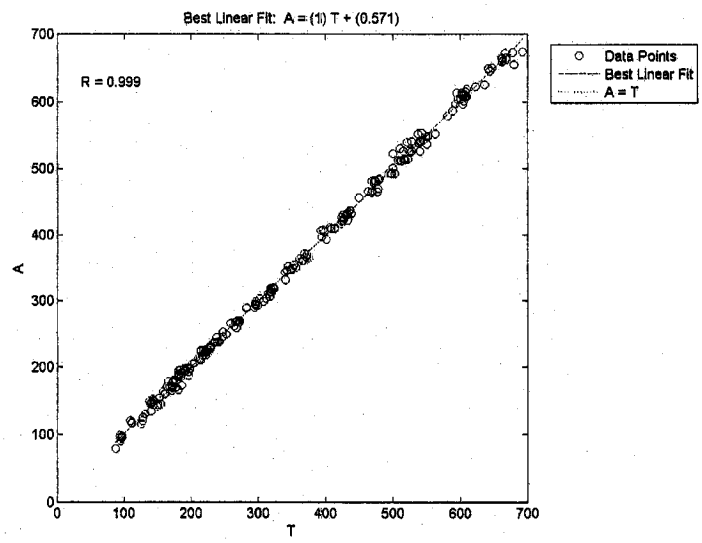

Figure D.3: Cycle Time (Network 2) Post Regression Plot, Model 0

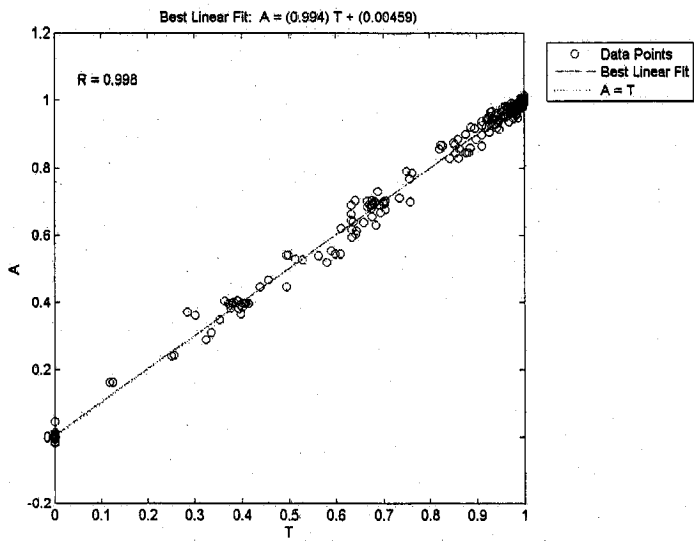

Figure D.4: Fill Rate (Network 3) Post Regression Plot, Model 0

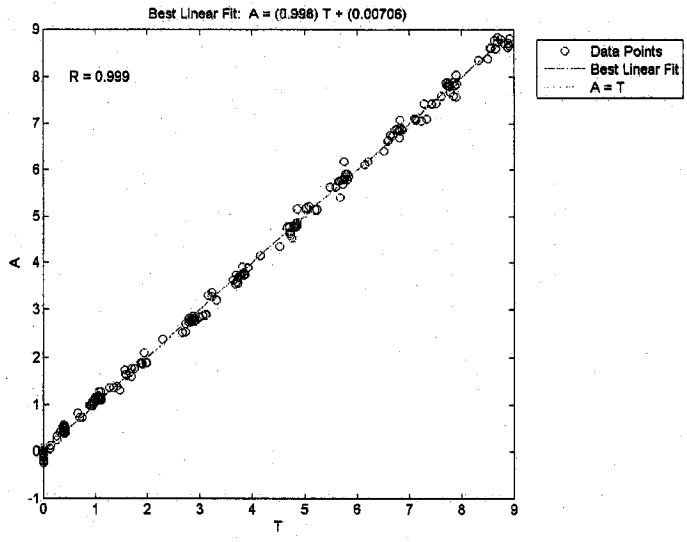

Figure D.5: Finished Goods Inventory (Network 4) Post Regression Plot, Model 0

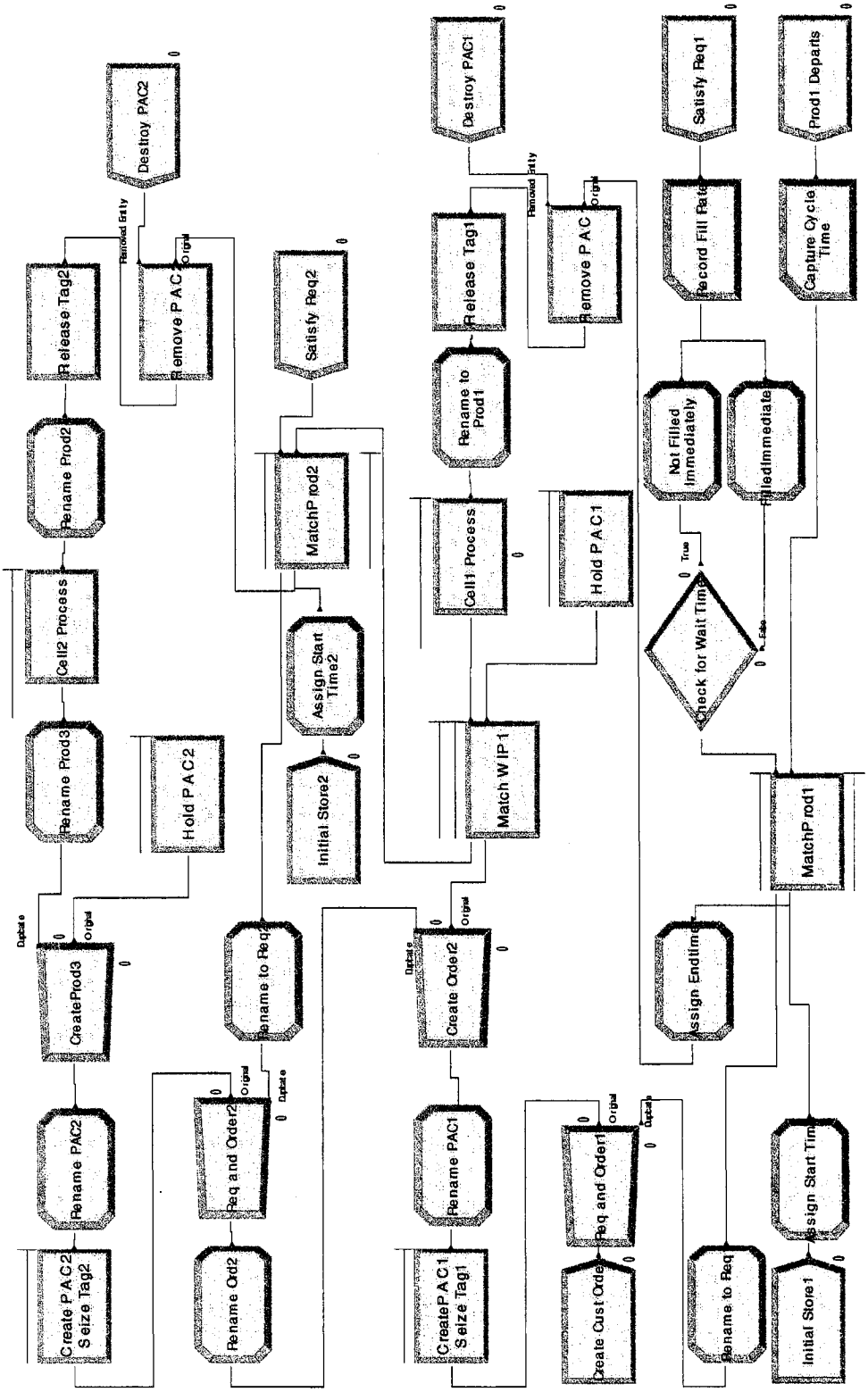

Figure D.6: Arena 9.0 Model for Example Model 0

Appendix E

Results for Model A

E.1 System Characteristics

Table E.1: Model A System Parameters

Product	Production Cell	Inputs (Part and Quantity)	Mean Processing Time (min)	Travel Time to Next Cell (minutes)
1	C3	P2 (Qty 1)	40	
2	C2	P3 (Qty 1)	36	5
3	C1	P4 (Qty 1)	42	5

E.2 Training Parameters

Minimum and maximum values were chosen (as in previous models), and the ranges for each parameter were then selected (Table E.2).

Table E.2: Ranges for PAC Parameters, Model A

	Low	Mid	High
z_1	0,2	3,6	7,10
k_1	4,6	7,9	10,12
z_2	0,2	3,6	7,10
k_2	2,4	5,8	9,12
z_3	0,2	3,6	7,10
k_3	1,3	4,6	7,10

The training dataset for this example model, using three levels per variable, would have generated 729 points. However, the following constraints were applied to the dataset:

$$k_2 \leq k_1 + z_2$$

$$k_3 \leq k_2 + z_3$$

This resulted in 677 design points in the training set. A MATLAB *m*-file (Appendix M) was used to count the number of valid combinations of parameters in the input space. For this model, it was determined that the networks would be valid for 1,043,196 parameter combinations.

Table E.3: Network Parameters, Model A

Network	Hidden Nodes
Customer Delay	20
Cycle Time	18
Fill Rate	15
FGI	18

E.3 Network Weights

Table E.4: Network Weights, Model A, Network 1

		Hidden Layer						
		Input Node i						
		0	1	2	3	4	5	6
Hidden Node j	1	4.4392	3.4284	0.3805	4.5945	7.0984	2.7504	1.6335
	2	3.184	-0.83215	0.39636	2.3277	-0.4	1.8804	-0.2286
	3	-0.68112	-1.3038	-0.58117	-0.08989	-0.15388	0.47078	-0.14398
	4	0.41921	0.76935	0.25707	-0.1475	-0.36981	1.7362	0.30187
	5	-2.176	-1.0833	0.11514	-2.4978	-0.0565	0.89442	0.00169
	6	-0.36008	-1.0274	-0.46365	-0.30088	0.52749	-2.1384	-0.4065
	7	0.75424	1.438	1.0681	0.48405	0.23315	-0.602	0.19694
	8	-4.6051	-3.4323	-0.38359	-4.5406	-7.4225	-2.6722	-1.7024
	9	-0.31833	-1.5125	0.16789	0.76562	0.06846	-0.1488	-0.22014
	10	0.48556	0.5804	0.13333	-0.44555	-0.2619	1.402	0.23321
	11	2.5182	-0.30902	0.28145	2.2451	-0.24072	1.3069	-0.08477
	12	3.7555	2.5827	0.00402	0.32445	-0.06059	0.13246	0.01711
	13	-0.42101	-0.92271	0.15405	0.9483	0.08401	-0.31519	-0.14246
	14	0.61583	-1.2707	0.08645	0.22005	0.07838	-0.02833	-0.00202
	15	3.0442	0.27934	0.23773	4.2844	-0.1458	1.1857	0.02249
	16	-1.4296	-0.90358	0.09333	0.96611	-0.13126	0.67125	-0.08227
	17	4.2313	3.4746	0.38976	4.747	6.6758	2.8985	1.5406
	18	2.7733	0.00646	0.22739	3.3613	-0.14525	1.0498	-0.01021
	19	0.80587	1.1864	0.0767	-0.52282	0.09562	-0.42745	0.10818
	20	-22.143	-10.184	-0.6482	-1.1999	-6.9771	-4.0068	2.4459

	Output Layer	
		Output
Hidden Node j	0	-2.3985
	1	-4.9725
	2	-1.7982
	3	3.1021
	4	5.9172
	5	-0.52935
	6	1.543
	7	0.9712
	8	-3.0584
	9	2.5588
	10	-6.1568
	11	5.6584
	12	-4.3237
	13	-4.461
	14	1.3416
	15	2.0375
	16	4.4485
	17	1.922
	18	-5.2004
	19	7.3349
	20	4.9831

Table E.5: Network Weights, Model A, Network 2

Hidden Layer								Output Layer	
		Input Node i							Output
Hidden Node j	0	1	2	3	4	5	6	0	1.7207
1	1.1945	-0.02605	0.69255	-2.2284	-0.2744	-0.72823	0.92582	1	2.5726
2	1.0377	0.05665	-0.08172	2.3335	0.04268	0.30341	0.00177	2	1.8232
3	-2.2534	0.04695	0.01328	0.28482	-0.03441	0.10012	-1.1761	3	-2.3311
4	1.0363	-0.07027	0.06443	1.4036	0.05506	0.34485	-0.12318	4	-3.4596
5	0.0519	0.2937	-0.3429	0.65812	-1.1776	-0.02945	-0.41473	5	-0.71393
6	0.54853	-0.82684	-0.27821	0.96563	2.2127	0.12609	-1.2489	6	-0.30247
7	0.37303	-0.18808	0.22452	0.43555	0.08918	0.00848	0.34515	7	1.7113
8	0.03334	0.62175	-0.33541	-0.48042	-0.07912	-0.05854	0.57861	8	0.40116
9	-1.183	0.05142	-0.00604	-0.86026	0.18257	-1.4706	0.05323	9	-1.7657
10	-0.10793	0.04855	0.02924	-0.59471	0.70718	0.2306	0.07782	10	-1.7144
11	-1.0423	0.06063	-0.13101	0.21436	-0.05472	-0.06079	-1.4101	11	1.0186
12	-1.2228	-0.11025	-0.77729	2.4741	0.22299	0.88503	-1.033	12	1.6881
13	0.21572	-0.59113	0.5188	0.24599	1.9995	0.06142	-0.16085	13	-0.47911
14	2.2237	0.85944	-0.35142	1.6519	0.91149	-0.04952	0.16982	14	0.42217
15	-1.6712	-0.06795	-0.00788	0.427	-0.17758	1.3419	-0.18688	15	1.8036
16	1.3974	-0.41454	0.4033	-1.9568	-0.33422	-0.33282	0.2751	16	-1.7178
17	-0.36774	0.56486	-0.13665	-0.6048	-1.4397	-0.127	0.61778	17	-1.2496
18	0.27276	0.02484	-0.09856	0.95957	-0.16221	0.14516	0.42329	18	-1.627

Table E.6: Network Weights, Model A, Network 3

Hidden Layer								Output Layer	
		Input Node i							Output
Hidden Node j	0	1	2	3	4	5	6	0	11.667
1	0.86113	1.6653	0.17185	0.51991	-0.10498	1.4656	0.02107	1	-6.865
2	-0.44676	-2.2055	-0.26191	-0.70121	0.14752	-2.0114	-0.05006	2	-2.48
3	1.4516	-0.29586	-0.11976	-1.4312	-11.104	-0.92481	0.53002	3	-0.02642
4	2.0927	2.2227	0.15551	0.4712	-0.10121	1.5099	0.029	4	3.8703
5	5.3435	3.8814	3.5518	3.5861	2.6197	0.54417	3.4527	5	0.83054
6	2.6093	-1.7652	0.23519	4.5781	-0.16235	-0.26153	0.03291	6	3.4515
7	3.4659	4.3298	-0.00526	0.48176	0.0193	-0.02643	-0.02114	7	2.2644
8	5.5015	4.0372	3.8284	3.5675	2.6923	0.56739	3.5307	8	-0.81238
9	-3.4681	1.6019	-0.16725	-2.256	0.2983	-3.6151	-0.11793	9	2.8639
10	2.4564	-1.5885	0.24817	4.6053	-0.14664	-0.23966	0.03618	10	-3.4727
11	-3.3535	1.5956	-0.15524	-1.9944	0.27659	-3.2609	-0.08904	11	-3.406
12	-7.3922	-5.0971	-0.26648	-4.9992	0.03083	0.01271	0.18043	12	-0.37844
13	-0.03113	-2.0004	-0.04778	0.21692	0.0284	-0.31227	0.02539	13	-5.6991
14	0.60806	1.9949	0.02553	-0.81294	-0.02381	-0.02446	-0.03695	14	-9.815
15	-1.0362	-1.7999	-0.02484	1.1391	0.04728	0.15195	0.03727	15	-5.6882

Table E.7: Network Weights, Model A, Network 4

Hidden Layer

Hidden Node j	Input Node i 0	1	2	3	4	5	6
1	-1.4896	-0.73649	-0.22921	-2.0315	0.60609	1.7444	-0.82779
2	2.4563	-1.8335	0.07427	1.7821	-0.05746	-0.03851	-0.1609
3	-0.98227	-1.8752	0.01964	0.60605	0.12772	0.24988	0.0047
4	1.1492	-1.9762	-0.16996	-0.55419	0.12852	-0.14543	-0.05748
5	0.59942	-3.8913	-0.35974	-1.0884	0.21602	-0.33379	-0.11757
6	0.2808	-0.23829	-0.18129	-1.7171	-1.4712	0.18776	0.18316
7	-0.36359	0.56213	-0.28293	0.27804	-0.30748	0.67081	0.14485
8	2.5602	-3.122	-0.14839	-0.51518	0.17305	-0.29435	-0.08643
9	-3.0252	-0.01095	0.17251	-1.5827	0.04758	-2.9363	-0.02948
10	2.9325	0.63328	-0.02981	0.54854	-0.0463	1.4527	0.04826
11	2.9783	0.20207	-0.14389	1.2993	-0.08075	2.5711	0.05681
12	-0.41632	-0.44204	0.11508	0.66505	0.66734	-0.03446	-0.13168
13	0.41732	0.02149	-0.17666	-1.4619	-1.2641	0.15647	0.17046
14	-2.203	0.19939	-0.01843	-3.0655	-0.11975	-0.0991	0.05445
15	2.2643	-2.2499	0.04678	0.87129	0.03972	0.07415	-0.07719
16	3.6216	1.7965	-0.02835	0.46976	0.13312	0.00276	-0.00458
17	-2.4145	-0.19495	-0.01079	-2.7812	-0.12301	-0.15639	0.02545
18	1.5031	0.76358	0.21595	1.9519	-0.58458	-1.7164	0.79346

Output Layer

Hidden Node j	Output
0	-0.3172
1	1.8421
2	1.7746
3	-1.789
4	-3.0725
5	0.61149
6	-0.94976
7	0.82824
8	1.6312
9	2.3645
10	-2.0301
11	3.6654
12	2.1102
13	1.677
14	2.4711
15	-2.8476
16	-3.3378
17	-2.918
18	1.9262

E.4 Error Results

Table E.8: Error Results, Model A

Network	Minimum Observation	Maximum Observation	MSE	Mean Error	Mean Absolute Error
Delay	0	332.27	5.932043	0.018386	1.832769
Cycle Time	149.15	1305.92	66.842247	-7.20E-03	6.148351
Fill Rate	0	1	0.000142	0.000003	0.009179
FGI	0	8.579	0.002883	0.000171	0.039592

E.5 Optimization of System

The training dataset was scanned to find all the points which, after one simulation each, satisfied all of the constraints. There were 24 points in this set, and the top 10 results, sorted by Finished Goods Inventory (FG Inv) are shown in Table E.9. The best point in this set had an average finished goods inventory of 4.021 units.

Table E.9: Points in the Training Data Set which Satisfy Constraints, Model A

Point Number	z_1	k_1	z_2	k_2	z_3	k_3	Cust Delay	Cycle Time	Fill Rate	FG Inv
239	6	8	1	4	2	6	9.4	297.4	0.915	4.021
6	8	9	0	2	1	2	13.4	201.1	0.908	4.709
471	8	6	0	6	1	7	3.5	234.9	0.945	4.927
279	7	12	3	6	0	5	5.7	295.1	0.932	4.929
477	8	10	0	8	2	9	3.6	287.2	0.951	5.262
87	8	11	0	4	3	1	0.8	278.6	0.983	5.424
270	8	12	1	5	2	4	4.0	296.6	0.966	5.880
3	8	5	2	4	1	1	1.6	236.4	0.979	6.087
60	9	11	0	9	2	2	3.1	254.5	0.974	6.236
81	9	5	0	3	3	1	0.5	284.7	0.989	6.314

Table E.10: Top Results from Neural Network Evaluation of Most Points, Model A

						Network Results				Simulation Results (Avg. of 20 Replications)			
z_1	k_1	z_2	k_2	z_3	k_3	Cust Delay	Cycle Time	Fill Rate	FG Inv	Cust Delay	Cycle Time	Fill Rate	FG Inv
5	9	2	8	2	1	9.8	284.8	0.901	3.259	8.0	287.1	0.908	3.310
5	8	2	8	2	1	9.4	286.6	0.904	3.260	8.2	285.5	0.904	3.276
5	10	2	8	2	1	10.2	283.2	0.904	3.260	8.6	287.8	0.900	3.289
5	6	2	8	2	1	9.0	291.5	0.912	3.261	9.1	287.0	0.897	3.268
5	7	2	8	2	1	9.1	288.8	0.910	3.261	7.6	287.9	0.910	3.314
5	10	3	8	1	1	9.1	279.0	0.900	3.264	6.9	286.6	0.916	3.365
5	8	2	7	2	1	10.9	287.1	0.903	3.265	7.7	288.0	0.907	3.314
5	10	2	3	2	1	15.5	276.6	0.904	3.268	9.1	283.4	0.899	3.264
5	5	2	7	2	1	10.0	295.4	0.910	3.269	8.0	286.1	0.901	3.284
5	4	2	6	2	1	10.5	298.6	0.912	3.269	9.6	286.2	0.898	3.281
5	7	2	7	2	1	10.4	289.3	0.909	3.269	8.0	288.6	0.906	3.305
5	9	2	5	2	1	14.1	284.1	0.904	3.270	8.1	287.6	0.907	3.311
5	6	2	7	2	1	10.1	292.0	0.911	3.272	7.2	287.5	0.910	3.312
5	8	2	6	2	1	12.2	286.9	0.907	3.272	8.8	288.2	0.902	3.292
5	10	2	2	2	1	14.5	270.5	0.909	3.275	8.8	280.2	0.902	3.286
5	9	2	4	2	1	14.7	281.7	0.909	3.278	8.6	287.9	0.906	3.320
5	7	2	6	2	1	11.6	288.9	0.912	3.280	8.5	288.2	0.904	3.294
5	4	2	5	2	1	10.3	296.2	0.916	3.280	9.1	283.7	0.902	3.269
5	5	2	6	2	1	10.7	294.7	0.913	3.281	7.7	286.9	0.911	3.329
5	8	2	5	2	1	13.3	285.6	0.911	3.282	7.4	287.8	0.910	3.320

Appendix F

Results for Model B

F.1 Network Parameters for Case I: No setups or travel times

Table F.1: Model B System Parameters

Products	Production Cell	Inputs (Part and Quantity)	Mean Processing Time (Minutes)
1	2	2 (Qty 1), 3 (Qty 1)	45
2	1	4 (Qty 1)	24
3	1	5 (Qty 1)	18

Table F.2: Networks for Model B, Case I

Network	Description	Hidden Nodes
1	Average customer delay time	20
2	Average fill rate	12
3	Average finished goods inventory	1
4	WIP Product 2	18
5	WIP Product 3	18
6	WIP Product 4	15
7	WIP Product 5	15

F.2 Network Weights, Case I

Table F.3: Network Weights, Model B, Case I, Network 1

Hidden Layer								Output Layer	
		Input Node i							Output
		0	1	2	3	4	5	6	0	-0.088304
Hidden Node j	1	-6.353	-0.84561	-0.34064	0.1999	0.17141	-3.7994	0.2926	1	3.0545
	2	-2.6298	-1.3047	1.3856	-0.60552	1.2981	0.89175	0.42609	2	-0.44065
	3	-3.7079	0.076976	-0.57646	-1.4455	-0.59839	-0.30566	-0.25954	3	11.924
	4	-0.95682	-0.93844	0.47918	-0.17031	-0.01125	-0.21069	-0.05089	4	5.1728
	5	-3.5141	-1.3929	-3.0047	-1.3051	1.2521	-3.4767	3.9468	5	0.18932
	6	0.32823	0.00361	-1.2708	0.17519	0.5565	-1.0215	1.1095	6	2.2657
	7	-0.22542	-1.1541	0.61881	-0.15299	0.06436	-0.26819	0.025787	7	-2.6509
	8	2.4352	0.92652	0.031937	0.49586	0.32524	1.346	0.093504	8	5.5983
	9	-0.17679	-0.10969	1.3083	-0.18253	-0.514	1.0031	-0.85916	9	2.372
	10	3.1981	0.14319	-0.5534	-1.8985	-4.799	1.7701	0.93386	10	-0.13668
	11	1.3547	0.50324	0.32094	0.58138	2.7049	0.219	0.050362	11	1.4834
	12	4.2097	0.49216	-0.3958	0.6233	0.44353	0.81189	2.7308	12	1.6337
	13	-1.001	-0.49437	3.0681	4.3557	1.8403	2.8355	0.048091	13	0.12499
	14	2.9297	0.77712	0.66469	0.97253	0.57972	2.2748	0.21655	14	-6.8632
	15	8.0349	1.1266	-0.44376	4.0799	0.4913	-0.24328	0.009125	15	-4.5464
	16	-5.5964	-0.49131	0.29354	-0.34211	-0.13572	-0.9798	-2.4943	16	5.7997
	17	-1.2711	-0.44708	-0.48052	-0.41584	-2.4439	-0.45437	-0.18073	17	1.6194
	18	5.0265	2.3368	0.1674	0.041334	-0.00718	0.081129	-0.01363	18	-8.541
	19	3.2766	-0.01477	0.8966	2.0392	0.8617	0.56367	0.27628	19	4.7828
	20	3.4184	0.75431	1.1973	1.1959	0.63368	3.3083	0.1603	20	3.0521

Table F.4: Network Weights, Model B, Case I, Network 2

Hidden Layer								Output Layer	
		Input Node i							Output
		0	1	2	3	4	5	6	0	9.7886
Hidden Node j	1	-25.095	1.9047	0.58556	-23.416	-2.1441	-1.4131	1.3025	1	5.5285
	2	-24.02	1.7272	0.50332	-22.484	-2.072	-1.2863	1.228	2	-5.6388
	3	12.419	5.1868	0.054104	0.21196	-0.32036	4.5467	0.41714	3	-10.29
	4	-4.2851	-6.7725	1.1858	-9.6529	2.1166	-0.66161	0.10179	4	0.073112
	5	-0.70707	-3.4465	-1.0247	1.7082	-0.16561	-0.08261	-0.65222	5	23.779
	6	-0.59054	-3.3788	-1.386	1.3166	-0.18361	-0.18914	-0.84527	6	-22.56
	7	-0.53149	-3.421	-1.7076	1.0652	-0.17793	-0.2799	-0.98759	7	8.9623
	8	-9.0671	-1.086	-0.54471	-0.3577	0.23971	-6.3073	0.38856	8	-3.3621
	9	2.7208	3.0591	-0.0381	0.95717	0.064584	-0.1074	-0.12149	9	1.5178
	10	-8.5443	-2.1981	1.1092	0.22157	1.1181	-6.0768	-2.751	10	-0.33599
	11	-0.82567	-3.6664	-0.85141	2.1212	-0.15156	-0.01149	-0.54031	11	-9.6519
	12	-2.058	-2.0379	-0.04813	0.43515	0.028877	-0.05912	-0.0693	12	-3.3205

Table F.5: Network Weights, Model B, Case I, Network 3

Hidden Layer								Output Layer	
		Input Node i							Output
		0	1	2	3	4	5	6	0	-6.2224
Hidden Node j	1	3.5543	-0.45183	-0.07682	1.7356	0.38184	-0.8041	-0.10332	1	9.0235
	2	-3.7691	2.5183	-0.10476	-0.25749	0.059023	-1.1935	-0.10938	2	2.0729
	3	-0.74226	-1.419	-0.00693	0.43621	-0.44141	-2.603	2.7732	3	-3.32
	4	1.6417	4.4363	-0.11165	0.26409	0.16609	0.10412	0.040014	4	0.33477
	5	-2.9786	0.58791	-0.09672	-1.8266	-0.33959	-1.1332	-0.26463	5	-2.1379
	6	2.8578	0.018234	-0.09149	2.5831	0.47689	-0.02076	0.19767	6	-5.2909
	7	2.8641	-0.88686	0.1948	-0.07392	-0.11637	3.2867	0.13594	7	-4.6857
	8	-0.78889	2.1162	-0.01813	0.012256	0.005658	-0.27253	-0.05379	8	2.1403
	9	0.7013	1.3839	-0.01316	-0.41423	0.42609	2.6253	-2.7081	9	-3.3894
	10	2.8527	-0.91395	0.14117	-0.10772	-0.12213	2.7584	0.13719	10	6.9065
	11	3.3011	0.49557	-0.16284	3.0521	0.51589	-0.08692	0.26441	11	2.533
	12	-3.8781	0.50914	0.081369	-1.4305	-0.36436	1.3733	0.24441	12	5.4788

Table F.6: Network Weights, Model B, Case I, Network 4

Hidden Layer								Output Layer	
		Input Node i							Output
		0	1	2	3	4	5	6	0	1.3208
Hidden Node j	1	-2.4452	-0.016	0.64548	-0.0555	-0.02451	-0.53902	-3.5844	1	-0.33689
	2	1.4918	-0.06975	0.35192	-0.91966	-0.81568	-0.16256	-0.28639	2	4.3061
	3	2.0624	0.19418	-0.96913	-0.41997	3.546	0.57401	-0.41527	3	1.1613
	4	2.9513	-0.11104	0.23353	-0.08686	-0.04048	2.8468	-0.00943	4	6.6307
	5	-0.42993	-0.01407	0.83182	0.05141	0.075483	0.14471	-0.18207	5	-3.1032
	6	3.0084	-0.20693	0.68988	-0.1558	-0.14352	3.2101	0.21482	6	-1.5832
	7	-3.3473	0.071963	-0.12249	0.25369	0.098447	-2.539	0.10264	7	7.7309
	8	-0.582	-0.06862	0.44691	-1.1559	-0.69113	-2.161	-0.46169	8	-0.5173
	9	0.55968	1.851	5.8494	-3.9987	-5.0813	1.5319	5.9797	9	-0.023685
	10	0.47276	0.009903	0.28215	0.093594	0.10934	-0.04127	-1.1151	10	-4.7967
	11	-5.4099	0.69561	0.54583	-3.6452	-5.0056	0.29926	-0.33268	11	-0.079291
	12	0.13292	0.086969	-0.00601	0.93351	-0.71106	-0.14839	-0.04117	12	1.1175
	13	2.312	-0.00644	-0.01413	1.88	0.23126	0.1637	-0.05534	13	-5.0769
	14	2.2686	0.000937	-0.05957	2.6237	0.1405	-0.08933	-0.02266	14	2.7143
	15	-0.43794	0.049235	-0.16959	1.0593	0.25977	-0.66426	0.10585	15	2.8787
	16	-0.03003	0.012157	-0.32393	-0.05519	-0.11748	-0.05559	0.46817	16	-13.977
	17	-1.8059	0.058991	-0.41351	0.95775	0.711	0.63934	0.44416	17	3.707
	18	-1.9402	-0.17305	0.87774	0.45995	-3.4921	-0.57254	0.27961	18	1.2391

Table F.7: Network Weights, Model B, Case I, Network 5

Hidden Layer								Output Layer	
		Input Node i							Output
	0	1	2	3	4	5	6	0	0.16817
Hidden Node j 1	1.1851	-0.10314	-0.68329	-0.23266	2.3748	-0.22205	-0.17722	1	3.4468
2	-0.25147	-0.05491	0.35885	-0.52355	-0.09606	-1.5951	-0.06947	2	-1.2175
3	-0.4038	-0.04015	0.67339	0.042902	0.051573	0.19484	-0.09453	3	-5.328
4	-4.3866	-0.10146	0.31254	-3.5585	-0.36748	0.33346	-0.28727	4	-2.0095
5	-1.3859	-0.05891	-0.24618	0.050802	-0.05378	0.78378	-1.1586	5	-2.9776
6	5.284	0.10146	-0.24194	3.564	0.28721	-0.43793	0.19069	6	-4.4416
7	0.065314	-0.02093	0.061241	-0.80114	-0.18305	1.3231	0.044652	7	3.8718
8	-0.36901	0.036392	-0.1791	1.5192	0.34816	-1.5536	-0.05585	8	1.2013
9	1.6874	-0.03044	-0.56525	-0.44099	0.96655	0.006224	-0.88627	9	2.3505
10	1.6672	0.013683	0.82196	0.26214	-0.08005	-0.41675	-0.73279	10	1.5645
11	1.4497	-0.05087	-0.43103	-0.06573	-0.51454	0.83506	-0.08211	11	-3.0706
12	1.3708	0.019217	1.1998	0.050736	0.009754	-0.62068	2.5746	12	-0.18016
13	-1.4182	-0.06349	0.34404	0.044619	-0.34139	-1.2941	-0.01942	13	3.8319
14	-2.2215	0.023878	0.13201	0.16213	-0.03211	-3.0862	-0.00195	14	-1.3669
15	-0.98644	-0.0795	0.28026	0.22888	0.084536	0.89357	0.035902	15	4.4399
16	1.3003	0.049399	-0.09281	0.042352	0.027054	-0.7383	1.5581	16	-1.9411
17	-1.2356	0.19069	0.95827	0.44737	-2.8808	0.26763	-0.05979	17	1.1938
18	-1.3568	0.063219	0.55259	0.21606	-1.4903	0.17586	0.45876	18	4.875

Table F.8: Network Weights, Model B, Case I, Network 6

Hidden Layer								Output Layer	
		Input Node i							Output
	0	1	2	3	4	5	6	0	-0.86402
Hidden Node j 1	-2.0375	-0.11331	0.19327	0.094682	-3.4622	0.15975	1.0445	1	6.2658
2	-0.15425	-0.18621	1.2934	0.08219	-0.18247	0.41453	-1.1795	2	-5.6529
3	3.3233	-0.19185	0.95794	0.049638	-0.10616	1.1087	2.0902	3	2.6774
4	2.2041	-0.03204	-0.17903	-0.03416	-0.81675	-0.37957	2.6745	4	5.6394
5	-5.0019	-0.31042	-7.6505	-10.285	-3.8979	2.3888	1.3893	5	-0.19571
6	2.7084	0.97663	4.858	8.7849	2.0947	-1.4455	-1.9207	6	-0.21943
7	-1.3526	-0.21596	-0.37203	0.24225	-1.0989	0.17996	0.15573	7	-3.5652
8	-0.44308	0.34223	-1.3532	-0.19324	0.1644	-0.47954	1.7869	8	-7.6401
9	-3.3334	-1.4815	-0.1872	-1.6745	6.3055	-1.0683	-1.8089	9	-0.24397
10	-0.6729	-0.17262	0.7843	-0.00393	-0.0344	1.5135	0.45858	10	3.2324
11	-2.6846	0.043555	0.3412	-0.05343	-2.281	-0.0325	-0.01571	11	-6.603
12	2.6604	0.26551	-0.35939	-0.12138	4.3256	-0.34471	-1.8201	12	3.0064
13	0.85317	-0.79776	2.0964	0.4562	-0.36208	0.9255	-2.6421	13	-1.8941
14	2.1248	-0.05865	-0.29708	0.017334	-0.64376	-0.33598	3.243	14	-4.7121
15	0.43653	0.14502	-0.59367	0.1829	0.061066	-1.9926	-0.07108	15	2.3696

Table F.9: Network Weights, Model B, Case I, Network 7

		Hidden Layer						
		Input Node i						
		0	1	2	3	4	5	6
Hidden Node j	1	5.6424	-0.05573	-0.29414	0.061543	5.2672	0.17561	-1.0883
	2	7.1278	-16.89	-3.9022	2.7625	-0.33041	3.8221	2.7825
	3	-1.3097	-0.09808	0.6338	-0.02418	-0.22949	0.021562	-1.3422
	4	-3.9591	2.1881	-5.7947	-5.4599	9.6218	7.111	11.383
	5	-1.4544	-0.4089	-0.53202	0.41626	-0.63194	0.034173	-1.6434
	6	1.0219	0.15129	-0.01518	-0.01728	-1.5792	-0.07269	2.083
	7	0.30587	-0.28528	0.3706	0.30821	-0.28832	0.48775	2.7524
	8	-0.62421	0.16567	-0.24935	-0.19741	0.067175	-0.11557	-0.99777
	9	-3.2574	2.4587	-0.14228	-1.4566	1.7168	-0.27582	3.2658
	10	-0.1322	-0.01281	0.38154	-0.12241	-0.15257	1.0476	-0.66304
	11	1.0355	0.13878	-0.07864	-0.04312	-1.2499	-0.12528	1.7016
	12	2.2323	0.044992	-0.18558	-0.08982	-0.48472	-0.08318	3.1145
	13	-0.07727	-0.02745	-0.16412	0.083861	0.30313	-0.28283	0.86466
	14	2.0058	-0.41714	1.0491	9.6648	4.5889	-0.18585	-5.3019
	15	2.3917	0.078346	-0.20534	-0.14616	-0.72027	-0.11482	2.7858

	Output Layer	
		Output
Hidden Node j	0	15.311
	1	2.1271
	2	0.087936
	3	-6.5213
	4	-0.098313
	5	-2.6025
	6	9.1523
	7	-2.1265
	8	-11.777
	9	0.42361
	10	-4.3239
	11	-14.568
	12	-12.536
	13	-12.638
	14	-0.11591
	15	14.912

F.3 Error Measurements, Case I

Table F.10: Error Measurements for Trained Networks, Model B, Case I

Network	Minimum Observation	Maximum Observation	MSE	Mean Error	Mean Absolute Error
Delay	0.01	383.2	31.890268	0.014512	4.264647
Fill	0	0.999	0.00041	1.00E-06	0.015434
FGI	0	8.145	0.011735	-0.000009	0.077397
WIP 2	0.245	11.28	0.006622	0.000023	0.063497
WIP 3	0.174	11.192	0.007107	0.000023	0.06415
WIP 4	0.105	1.098	0.001792	-0.000002	0.032153
WIP 5	0.381	1.511	0.00199	-0.000038	0.033288

F.4 Post Regression Plots, Case I

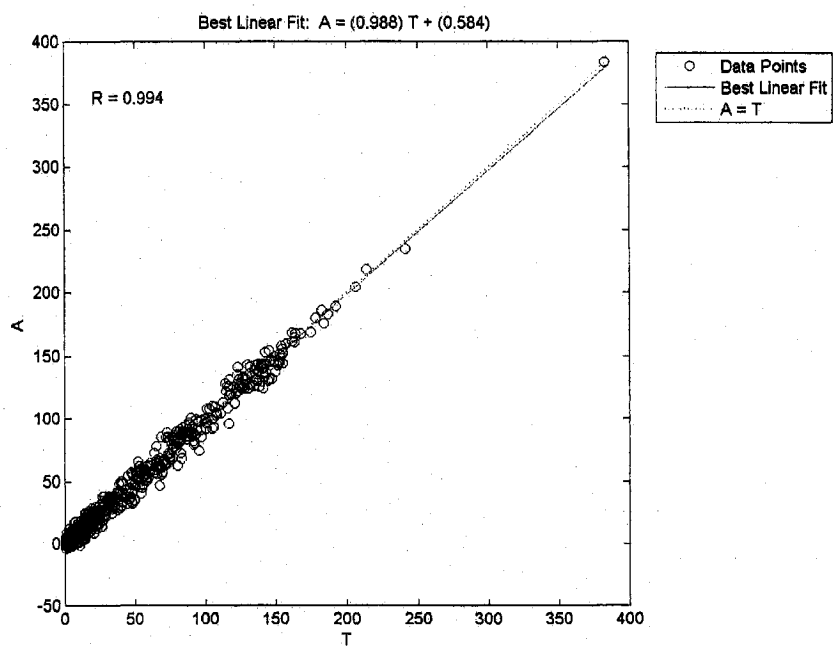

Figure F.1: Customer Delay Time (Network 1) Post Regression Plot, Model B

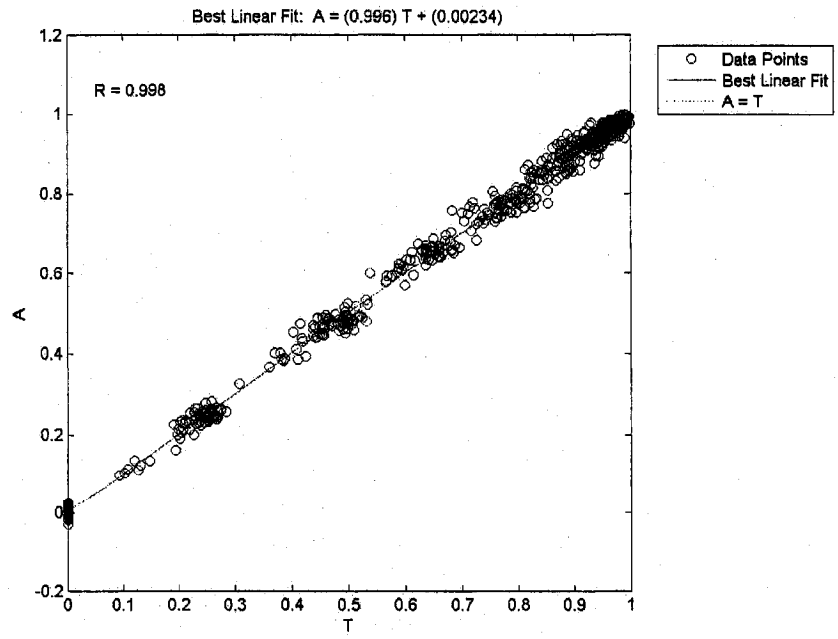

Figure F.2: Fill Rate (Network 2) Post Regression Plot, Model B, Case I

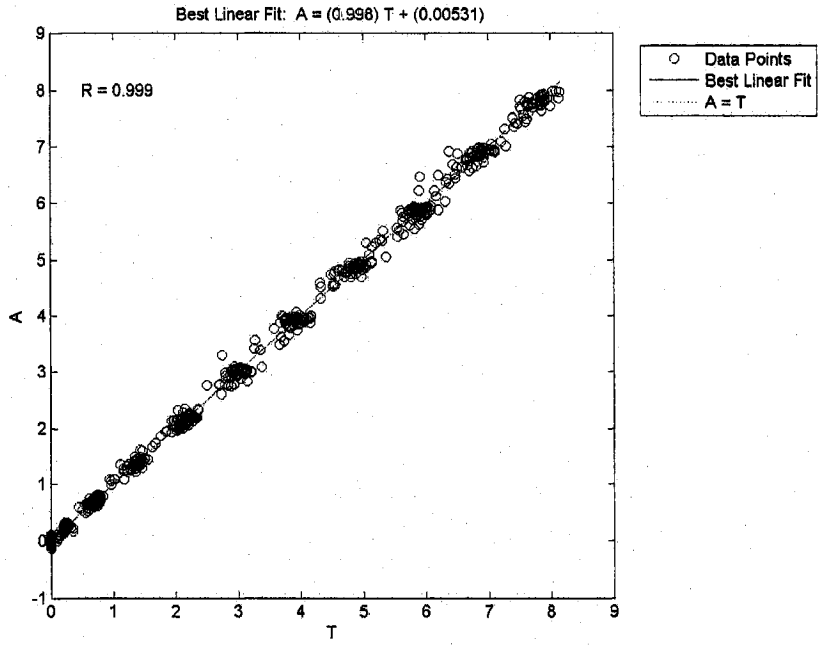

Figure F.3: Finished Goods Inventory (Network 3) Post Regression Plot, Model B, Case I

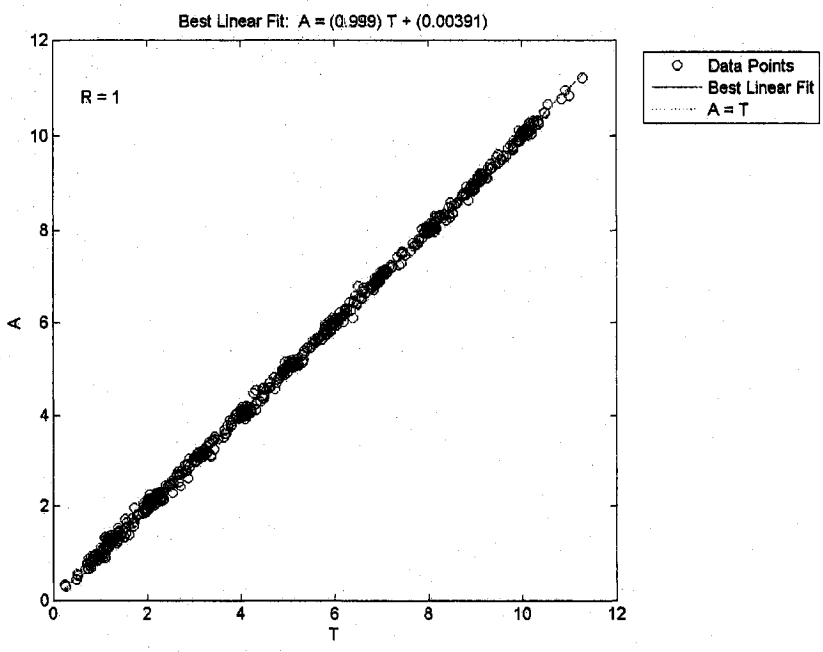

Figure F.4: Work-in-Process, Product 2 (Network 4) Post Regression Plot, Model B, Case I

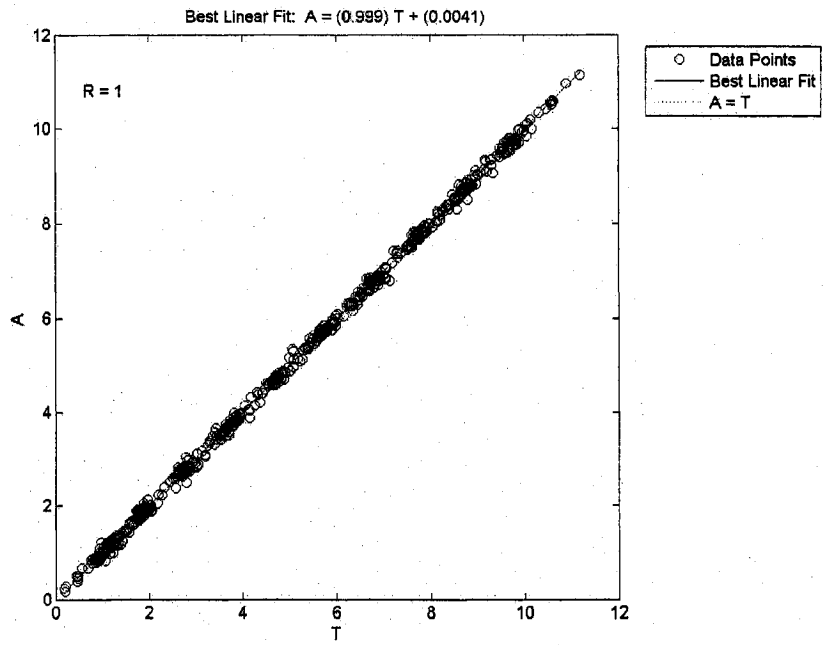

Figure F.5: Work-in-Process, Product 3 (Network 5) Post Regression Plot, Model B, Case I

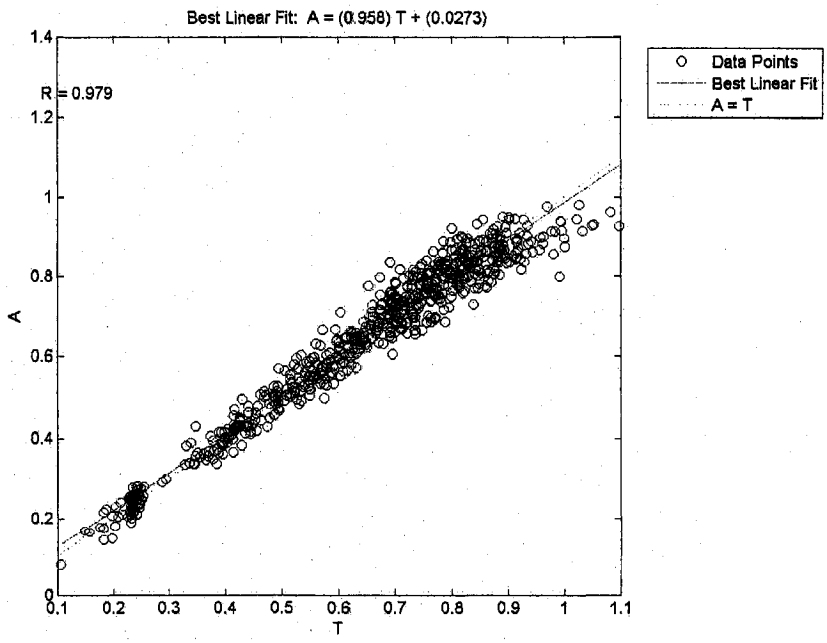

Figure F.6: Work-in-Process, Product 4 (Network 6) Post Regression Plot, Model B, Case I

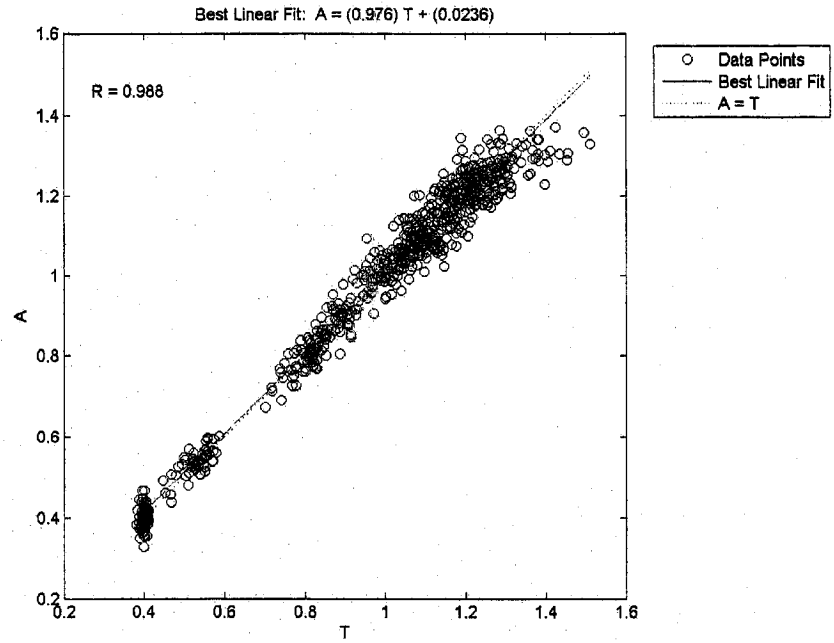

Figure F.7: Work-in-Process, Product 5 (Network 7) Post Regression Plot, Model B, Case I

F.5 Network Parameters for Model B, Case II (Setup and travel time)

Table F.11: Networks for Model B, Case II

Network	Description	Hidden Nodes
1	Average customer delay time	30
2	Average fill rate	12
3	Average finished goods inventory	12
4	WIP Product 2	18
5	WIP Product 3	24
6	WIP Product 4	20
7	WIP Product 5	18

F.6 Network Weights, Case II

Table F.12: Network Weights, Model B, Case II, Network 1

Hidden Layer

Hidden Node j	Input Node i 0	1	2	3	4	5	6
1	11.177	7.7347	-2.7542	15.393	-1.3193	-0.42584	-1.5856
2	-0.60157	3.5459	-1.7008	0.81587	-8.9265	2.9698	-10.072
3	2.3828	1.2022	0.32947	1.0875	0.72577	0.54676	4.5045
4	-24.471	1.1995	-14.47	1.5276	9.2334	-20.155	-0.47545
5	0.46134	-1.6328	1.8139	1.9772	-1.0322	-0.55091	-0.69168
6	3.6987	3.8857	3.4463	4.8756	2.4668	7.9027	-2.1664
7	-0.41769	-0.73222	1.8367	1.1497	1.9475	0.44213	2.1452
8	1.0943	-1.8443	-0.93358	0.81297	2.1741	-3.2313	1.8177
9	1.118	-5.8025	12.109	8.6224	15.307	-0.71932	-11.329
10	0.24905	-1.6436	-0.33404	0.033343	-0.49092	0.074296	0.41887
11	3.5962	3.6286	3.3846	4.7818	2.392	7.4829	-2.0778
12	-0.08301	1.6839	0.30838	-0.14177	0.55353	-0.14524	-0.33729
13	-7.3901	-6.0086	-0.43939	-5.3483	15.897	-4.1731	-7.2059
14	-0.68976	20.089	-35.661	-16.045	7.2476	6.4617	1.0905
15	1.9265	-1.4932	0.71756	1.7923	8.3629	9.7284	-0.78056
16	-2.6735	-3.2335	0.31108	2.1533	0.058403	-1.2099	0.034189
17	6.5283	0.44886	6.6077	7.1731	2.4709	-1.0189	0.69551
18	1.2924	0.062009	0.36326	-1.6682	-4.9787	5.1888	1.4507
19	2.2436	1.1238	0.30438	1.1082	0.6955	0.41421	4.1479
20	-21.518	-2.9855	-7.3602	-2.0945	17.986	-21.303	-6.4309
21	7.4355	6.2724	0.41612	5.4576	-16.322	4.3959	7.304
22	-4.9114	3.9869	-0.40268	-1.8401	-13.329	-11.836	3.0115
23	10.184	5.9487	-0.28959	-0.57239	-0.39981	1.4223	-0.08075
24	-4.7331	-1.5638	0.2695	-1.99	-0.15353	0.10599	0.10473
25	-1.0153	-0.05991	1.7855	1.4457	1.615	0.79769	1.216
26	0.43975	-1.6409	2.0131	1.8488	-1.0621	-0.51183	-0.74148
27	-6.3894	-0.13895	-2.4847	0.49041	1.5263	-6.1753	0.27831
28	2.4248	2.2337	1.2406	0.12944	-1.093	2.1185	-1.3066
29	11.86	3.6463	9.1898	14.05	10.831	9.9121	-9.7441
30	-2.3063	-2.1974	-1.2057	-0.40377	0.88159	-1.7081	1.1924

Output Layer

Hidden Node j	Output
0	4.0205
1	0.15449
2	-0.14566
3	3.3446
4	-0.40201
5	2.6681
6	-3.8177
7	-0.55579
8	-0.40857
9	-0.10268
10	5.8073
11	4.005
12	5.6283
13	-3.8496
14	-0.088706
15	-0.20891
16	0.6823
17	0.18667
18	-0.2451
19	-3.6562
20	0.17685
21	-3.7856
22	-0.16796
23	-11.437
24	4.2222
25	0.661
26	-2.6529
27	0.83282
28	4.8892
29	-0.11043
30	5.21

Table F.13: Network Weights, Model B, Case II, Network 2

Hidden Layer

Hidden Node j \ Input Node i	0	1	2	3	4	5	6
1	0.54381	0.90126	0.28246	-0.74769	0.95132	0.52922	-0.42029
2	-1.9131	1.5212	0.48193	0.84966	-2.6789	-0.77376	1.7215
3	-1.8794	3.7883	2.9195	1.0588	-2.4225	-4.2332	0.047233
4	-0.44605	-0.86307	-0.28044	0.66409	-0.76451	-1.2115	0.3988
5	-2.0124	-0.39788	0.000262	-1.9189	0.14139	0.28059	0.017938
6	0.83634	1.2331	0.55355	0.074766	0.8619	-2.9964	-0.70519
7	2.3761	0.19894	0.092909	2.1268	-0.54763	0.13201	0.11139
8	-1.4437	-2.0634	0.008585	1.1651	0.49027	-1.6904	-0.47543
9	2.195	0.56784	-0.07152	1.6757	0.12624	-0.51968	-0.11112
10	0.39417	0.92899	0.30552	-0.71046	0.63679	1.7762	-0.38813
11	-1.8211	-2.8566	-0.26281	-1.1233	0.20123	-1.0486	0.032596
12	-3.87	-3.8153	0.083307	0.61466	0.12466	0.48045	-0.03874

Output Layer

Hidden Node j	Output
0	-0.15946
1	-5.2142
2	0.26514
3	-0.13946
4	-11.815
5	9.8587
6	0.40324
7	4.6312
8	-0.69389
9	7.8798
10	-6.3858
11	-0.70562
12	-1.5801

Table F.14: Network Weights, Model B, Case II, Network 3

Hidden Layer

Hidden Node j \ Input Node i	0	1	2	3	4	5	6
1	0.14404	-0.93418	0.61771	1.3929	0.82761	-0.64619	-1.5973
2	1.5919	-0.20899	0.094118	-0.49203	0.022703	0.76712	0.007718
3	0.1719	-1.2373	0.21747	-0.23296	1.4868	-0.19476	-1.3635
4	0.77118	-1.2073	0.027042	-0.24292	0.028689	-0.12573	0.020271
5	-0.39593	-0.09691	-0.16329	0.063645	-0.07966	0.495	-0.15019
6	1.4362	-0.22624	0.049306	0.49008	-0.03929	0.60348	0.021527
7	1.5431	-0.55422	-0.07682	0.36561	1.0131	-1.8014	-0.7086
8	0.34596	0.17977	0.25482	-0.22566	0.1833	-0.09309	0.13497
9	-0.29712	1.103	-0.25993	0.000639	-1.3822	0.36546	1.4279
10	-0.19584	0.8998	-0.59647	-1.1526	-0.91806	0.60523	1.6528
11	1.0455	-0.36339	-0.08618	2.145	-0.17233	-0.279	0.04892
12	1.1131	-0.36704	0.12578	-0.08275	-0.0638	1.35	0.031403

Output Layer

Hidden Node j	Output
0	-4.2958
1	-4.3862
2	14.261
3	3.1817
4	-5.3541
5	-8.1402
6	13.17
7	0.81892
8	-6.8067
9	4.6335
10	-5.5117
11	-1.3521
12	-8.1255

Table F.15: Network Weights, Model B, Case II, Network 4

Hidden Layer

Hidden Node j	Input Node i 0	1	2	3	4	5	6
1	-1.1	-0.02726	-0.13054	0.57299	0.11022	-1.7607	0.088344
2	-0.23034	0.074466	0.12435	-0.36391	-9.9142	0.80389	9.4873
3	-0.45318	0.10855	0.029932	-0.5392	-10.338	0.89229	10.397
4	0.81671	-0.37774	0.66343	0.80534	0.57147	0.16248	-0.35161
5	0.26233	-0.17341	0.20792	-0.66046	0.51588	0.71947	1.1125
6	-0.96821	-0.07758	-0.23553	2.0644	0.88786	-1.4615	-1.6521
7	2.5501	1.2747	0.096313	-0.62543	2.8181	-5.7076	-2.3232
8	-4.7713	1.6973	1.8058	2.733	4.0238	-0.37095	-8.7565
9	-1.0228	0.07118	0.001837	-2.2743	-0.1379	0.41645	0.34689
10	1.3102	0.012256	0.088648	-0.72611	-0.07384	1.3915	-0.0519
11	-0.07337	0.047294	0.17234	-0.2387	-9.62	0.65723	8.7097
12	0.29109	-0.04097	0.69982	0.56731	-1.6488	0.45477	0.82168
13	0.43309	-0.07019	0.050211	-0.51705	0.10347	0.33923	0.87089
14	-1.9734	0.003392	0.027593	-0.35805	0.096171	-0.52629	-0.46618
15	0.77078	-0.38189	0.92303	0.22012	0.74013	0.071064	0.013864
16	-1.0913	0.034016	-0.01068	0.55905	0.000427	0.1645	-0.56319
17	-2.6914	-1.4288	-0.23022	0.82	-2.97	6.0466	2.6242
18	-0.12074	0.045825	-0.79668	-0.60959	1.6725	-0.67037	-0.89169

Output Layer

Hidden Node j	Output
0	-1.2631
1	-6.0317
2	-4.6225
3	2.5608
4	-1.0195
5	-2.2729
6	0.39973
7	-0.90296
8	0.046475
9	-1.1136
10	-8.9114
11	2.0255
12	2.7085
13	9.8667
14	7.2822
15	0.80841
16	9.5658
17	-0.82239
18	2.3747

Table F.16: Network Weights, Model B, Case II, Network 5

Hidden Layer

Hidden Node j	Input Node i 0	1	2	3	4	5	6
1	-0.81074	0.19339	-1.2937	-0.97335	-2.8826	0.04768	-0.88294
2	-2.0728	-0.27223	0.1649	0.75908	0.30153	0.82935	-0.48046
3	-1.3305	1.1884	-0.63651	-0.26166	-0.14347	-0.00657	-0.52093
4	-2.1565	-0.23442	-0.00069	0.19767	0.30644	1.6485	0.004436
5	1.1734	-0.02826	0.10046	1.0814	6.4037	0.1126	-5.1842
6	1.4827	0.010167	0.12722	2.4934	0.15431	-0.3793	-0.50673
7	1.8966	-0.05126	-0.05476	0.044824	-0.31105	-1.5346	-0.54121
8	-1.291	0.22554	-0.23606	-0.13453	1.1437	-0.97257	-0.49353
9	-0.42127	1.3978	0.1294	-0.75108	-4.804	-0.11376	3.4744
10	1.6259	-0.03232	-0.0622	-1.9678	-0.05768	0.22748	0.28405
11	-2.4927	-0.02281	-0.20067	-3.6868	-0.40451	0.96312	0.97663
12	-1.4452	-1.656	-0.18426	-0.29098	0.41156	-0.09889	0.30479
13	-0.6353	0.12525	-0.01942	-0.05819	-0.00495	-1.6811	0.003993
14	-1.0848	-0.42929	0.13022	2.5183	20.368	-1.2439	-21.561
15	0.82691	-0.07329	0.18241	0.95978	0.16882	0.30484	-0.46571
16	-0.8723	0.21555	-1.3743	-0.9739	-2.7488	-0.02358	-0.87616
17	0.99675	0.41717	-0.16857	-2.5033	-19.811	1.1598	21.08
18	0.62685	-1.7657	-0.11101	0.91363	4.9269	0.32982	-3.9868
19	-0.84162	1.9446	0.10405	-1.1134	-5.1783	-0.52767	4.6483
20	-1.5252	1.0857	1.1192	-0.49617	0.087184	-0.34159	-2.7736
21	1.2553	0.78456	0.30508	0.43111	-0.15078	-0.15781	-0.25017
22	0.25526	0.15611	-0.23643	0.1268	0.034418	-0.12226	0.65083
23	-1.4301	0.81195	1.0813	-0.46555	0.18309	-0.37072	-2.5267
24	0.48928	0.31035	0.15692	1.8604	0.54032	-1.6384	-0.43662

Output Layer

Hidden Node j	Output
0	3.8508
1	-1.6972
2	-8.6374
3	-1.076
4	8.272
5	-0.25402
6	4.9005
7	2.0476
8	1.6494
9	-2.3999
10	-3.6148
11	1.3507
12	1.9175
13	-4.6951
14	4.4511
15	-10.036
16	1.7581
17	4.5127
18	-4.009
19	-1.9115
20	1.265
21	5.6934
22	-5.3205
23	-1.6243
24	-0.54505

Table F.17: Network Weights, Model B, Case II, Network 6

Hidden Layer								Output Layer	
		Input Node i							Output
Hidden Node j	0	1	2	3	4	5	6	0	-6.2443
1	-1.8618	-0.35357	0.22557	1.2009	-11.294	1.1672	10.625	1	-5.2343
2	3.3477	-0.10752	0.63046	0.16342	-2.0706	0.36312	2.658	2	1.5023
3	0.21257	-0.29007	-2.3696	0.071211	3.265	-2.3633	2.6165	3	-1.4607
4	1.0763	0.053854	-0.07398	-0.33918	0.20081	-2.6382	0.54813	4	0.77823
5	0.13739	-0.94598	2.1011	1.0635	1.0019	-0.04233	-0.04251	5	-11.638
6	-3.4737	0.09464	-2.8125	0.36024	-0.64617	-3.0049	2.1675	6	-1.4252
7	-1.697	-0.34187	0.25121	1.2932	-10.916	1.1863	10.265	7	5.121
8	-0.03181	-0.14686	0.74775	0.34633	0.1339	0.14653	-0.95681	8	5.0693
9	0.27373	-0.17509	-2.487	0.19987	2.5845	-2.4043	2.342	9	1.7225
10	5.3109	0.028938	5.6694	-0.35833	-0.47475	6.0127	-4.6043	10	-0.74914
11	-4.6148	-0.28352	-1.3416	-0.34494	3.4178	3.1219	2.3673	11	0.54775
12	-0.15472	-0.94736	2.1317	1.2036	0.65343	0.070633	-0.36734	12	14.862
13	-5.6249	-3.8597	-1.2097	5.9686	9.6784	30.172	3.2173	13	0.14301
14	-0.10703	-0.00734	-0.16494	0.17179	3.0133	-0.1175	-1.5187	14	-1.1454
15	-6.4479	-1.608	-2.6676	-1.1586	1.8803	2.0495	1.3234	15	-0.73471
16	8.5815	2.3626	6.4066	19.704	27.773	5.1411	12.178	16	-3.7459
17	-0.47092	0.81608	-1.7565	-0.75091	-1.3108	0.0699	-0.13977	17	-4.876
18	1.1701	0.11196	-0.48479	-0.04001	1.1519	0.27017	0.72313	18	3.1941
19	0.27818	0.77713	-1.8796	-1.1156	-0.40177	-0.16784	0.64162	19	9.5821
20	-8.9416	-2.3776	-6.7029	-20.228	-28.767	-5.1393	-12.535	20	-3.7222

Table F.18: Network Weights, Model B, Case II, Network 7

Hidden Layer								Output Layer	
		Input Node i							Output
Hidden Node j	0	1	2	3	4	5	6	0	3.1815
1	-3.337	0.039856	0.002606	-0.10073	-3.3728	-0.13317	1.1794	1	-1.2685
2	0.36538	-0.04507	0.26837	0.36003	7.3951	0.000217	-7.4196	2	2.6372
3	0.054185	-0.01485	0.18894	0.10819	7.9533	-0.10555	-7.6695	3	-2.2795
4	-17.574	-5.9749	-13.662	3.3948	-4.6482	9.0076	8.566	4	6.1805
5	-2.954	0.00256	0.044107	0.001787	0.093417	0.057359	-1.7584	5	-4.6007
6	25.525	1.1992	19.335	-43.194	-10.093	-4.8812	9.1398	6	-3.4317
7	-1.1869	0.97679	-1.6208	-6.4474	-14.234	-1.3645	14.457	7	0.23153
8	4.5531	6.7574	-18.71	-12.036	-22.82	12.265	-19.562	8	0.13685
9	-1.8079	-0.66477	3.0471	0.62296	7.3729	-3.936	5.5603	9	0.17956
10	-1.7919	0.88352	1.7162	-0.32156	-5.1818	1.9879	-3.242	10	5.133
11	19.852	18.229	4.2442	48.522	-11.855	12.4	-45.81	11	-0.080123
12	-1.8963	0.90348	1.6971	-0.37454	-5.2978	2.0158	-3.3552	12	-5.0707
13	-25.054	-1.1508	-18.847	42.175	9.584	4.796	-8.9927	13	-3.4685
14	14.03	-2.4792	-2.5743	-9.0055	-18.04	-44.008	-4.5608	14	-0.058484
15	-3.4512	0.16235	-1.5978	0.33121	1.4739	-0.82096	2.0045	15	6.0624
16	-18.502	-0.88992	-9.9924	-0.59295	0.81814	-13.078	8.3237	16	-1.7605
17	-3.7321	0.22205	-1.8323	0.27134	1.4999	-0.89023	2.0374	17	-6.1366
18	-17.151	-5.8268	-13.201	3.2233	-4.6159	8.7986	8.2682	18	-6.2959

F.7 Error Measurements, Model B, Case II

Table F.19: Error Measurements for Trained Networks, Model B, Case II

Network	Minimum Observation	Maximum Observation	MSE	Mean Error	Mean Absolute Error
Delay	0.38	395.47	80.468268	-0.002416	7.172903
Fill	0	0.992	0.000989	-1.00E-06	0.024146
FGI	0	7.833	0.02518	-0.000432	0.114295
WIP 2	0.341	11.352	0.013097	0.000012	0.088943
WIP 3	0.256	10.022	0.01105	0.000426	0.081834
WIP 4	0.574	3.373	0.011469	-0.000004	0.079452
WIP 5	0.859	3.729	0.009737	-0.000251	0.071524

Appendix G

Results for Model C

Table G.1: Model C System Parameters

Product	Production Cell	Inputs (Part and Quantity)	Setup Time (Minutes)	Mean Processing Time (Minutes)	Move Time to Next Cell (Minutes)
1	2	3 (Qty 1)	--	90	--
2	3	4 (Qty 1)	--	60	--
3	1	5 (Qty 1)	25	25	15
4	1	5 (Qty 1)	25	25	10

Table G.2: Neural Networks for Model C

Network	Description	Hidden Nodes
1	Average customer delay time, Product 1	15
2	Average customer delay time, Product 2	15
3	Average fill rate, Product 1	15
4	Average fill rate, Product 2	15
5	Average finished goods inventory, Product 1	15
6	Average finished goods inventory, Product 2	15
7	WIP Product 3	15
8	WIP Product 4	15
9	WIP Product 5	15

Table G.3: Network 1 Weights, Model C

Hidden Layer												Output Layer	
		Input Node i											Output
		0	1	2	3	4	5	6	7	8	9	10	0	0.65565
Hidden Node j	1	-0.5403	-0.3278	-0.0017	0.02262	0.0006	-2.5667	0.017	2.1284	-0.0371	0.06669	0.08279	1	1.1356
	2	3.9606	3.2262	0.0186	-0.0345	-0.016	-0.1341	0.04498	0.13452	-0.0219	-0.0031	0.05356	2	2.606
	3	-2.2497	-1.4057	0.01428	0.00281	-0.0088	-3.3724	0.03516	2.6142	-0.0075	0.00836	0.05289	3	4.3318
	4	1.1268	0.60635	0.00838	-0.0126	0.00524	2.1647	-0.0057	-1.7797	0.0213	-0.0458	-0.071	4	3.4894
	5	-0.1737	0.01844	-0.0015	7.5E-05	-0.0011	-0.0051	8.7E-05	0.00077	-0.0004	-0.0024	0.00083	5	1.659
	6	3.6564	1.1695	0.05329	0.04425	0.2875	1.6904	0.10187	-2.3804	-0.0135	-0.309	-1.2124	6	0.79475
	7	-2.8164	-0.932	-0.0299	0.02151	-0.0952	-1.397	-0.0121	1.5273	-0.0066	0.09214	0.57746	7	2.7326
	8	6.6403	2.9405	-0.0166	-0.0373	-0.0091	4.5984	-0.0586	-0.9177	0.1479	0.18714	-0.7883	8	-0.1948
	9	2.1782	1.0171	-0.0086	0.10174	-0.1124	1.5263	0.11669	-0.7429	-0.0668	-0.0049	0.89246	9	0.82892
	10	10.582	0.7258	4.4611	-0.5624	0.02501	7.6739	0.69532	-4.7583	-0.287	0.05714	-0.231	10	-8.9574
	11	-2.2697	-1.0058	-0.0022	-0.0426	0.05752	-1.4251	-0.0563	0.89731	0.02931	0.01601	-0.4301	11	3.1482
	12	3.6688	4.0618	0.0505	-0.0206	0.04999	1.823	-0.0296	-1.2714	-0.0339	0.08791	-0.1266	12	-0.4686
	13	-9.9804	-0.7828	-4.7935	0.66741	-0.0333	-8.3537	-0.6799	5.0174	0.35992	-0.0659	0.23841	13	-2.6262
	14	4.5555	3.3746	0.00822	-0.0253	-0.0176	-0.2676	0.03469	0.20917	-0.0144	-0.0079	0.04624	14	-3.9325
	15	2.3114	1.8088	-0.0139	0.00327	0.00604	3.5159	-0.035	-2.6729	-0.0051	0.00187	-0.0295	15	3.4117

Table G.4: Network 2 Weights, Model C

Hidden Layer												Output Layer	
		Input Node i											Output
		0	1	2	3	4	5	6	7	8	9	10	0	0.19872
Hidden Node j	1	0.43359	0.04552	0.00979	-1.3979	-0.0785	0.01895	-0.0467	0.0214	-0.7958	-0.1384	0.63539	1	2.118
	2	-6.8338	0.0397	0.06619	-0.2846	-3.3004	0.13075	0.07531	-0.0491	-4.9954	-0.4022	3.4794	2	-1.8401
	3	-3.1814	0.01491	-0.0106	-2.7236	0.03826	-0.0277	-0.0214	-0.0931	-4.2111	-0.056	3.2835	3	-2.3589
	4	-1.1851	-0.1107	-0.0124	-0.2886	0.15615	-0.0111	0.09436	-0.0717	0.12084	0.26951	0.18224	4	2.6998
	5	0.63052	0.00516	0.00562	-0.8901	-0.0131	0.01187	-0.0089	0.02622	-1.1316	-0.0428	0.81636	5	-2.4574
	6	-2.4056	0.04037	0.00298	-2.1598	-0.0601	0.00866	-0.0306	0.02379	-0.9466	-0.1041	0.73903	6	3.8703
	7	-1.072	-0.0043	0.00828	-0.3785	0.01731	0.00341	-0.0073	0.08988	-1.6153	-0.0096	0.79501	7	4.3015
	8	3.1772	-0.0183	0.00904	2.5121	-0.0304	0.03579	0.02566	0.0673	4.0997	0.04098	-3.2591	8	-2.5477
	9	-2.1587	-0.0068	-0.0085	-1.1611	0.02818	-0.0106	0.00227	0.01217	0.48735	0.02093	-0.636	9	1.8719
	10	-1.7666	-0.2053	-0.0195	0.34314	0.29605	-0.0327	0.17251	-0.1044	0.42597	0.51046	-0.0406	10	-0.9623
	11	0.60634	0.00396	0.00397	1.5437	-0.0076	-0.0023	0.00042	0.0214	-0.335	-0.0097	0.20301	11	1.8929
	12	2.6477	-0.0023	0.03084	3.6455	-0.027	0.02049	0.00203	0.00436	-0.2396	0.00275	0.30721	12	0.52282
	13	1.0096	-0.006	-0.0103	0.66776	-0.0156	-0.006	0.02865	-0.1344	1.991	0.0398	-0.8045	13	2.2037
	14	1.1907	0.0015	-0.0099	1.3577	0.02475	-0.004	-0.0077	-0.0313	1.8771	0.00183	-1.551	14	1.7511
	15	7.4979	-0.0318	-0.0593	0.30613	3.0426	-0.1108	-0.0816	0.03612	4.6317	0.37328	-3.2928	15	-5.1054

Table G.5: Network 3 Weights, Model C

Hidden Layer												Output Layer	
		Input Node i											Output
		0	1	2	3	4	5	6	7	8	9	10	0	2.6399
Hidden Node j	1	-1.9643	-0.3615	0.00416	0.01165	0.04426	2.4784	-0.0074	-1.5812	0.01907	0.01085	-0.0359	1	1.3111
	2	-0.4539	-1.9854	-0.0394	0.0038	0.03537	2.3683	-0.0221	-1.8748	-0.0028	-0.0007	-0.038	2	3.658
	3	0.40889	2.1497	0.05466	-0.0106	-0.0205	-2.1453	0.02854	1.6043	-0.0451	-0.0085	0.0263	3	-3.3584
	4	0.17719	2.6916	0.07826	-0.0205	-0.0436	-2.4474	0.04219	1.9803	-0.0283	-0.0077	0.04363	4	2.7135
	5	5.3766	5.492	0.00981	-0.0451	0.01258	6.2546	-0.0038	-4.9453	-0.0209	-0.0452	-0.0062	5	0.93044
	6	7.1045	1.4148	0.24953	0.22655	-0.0462	4.6084	0.24386	2.4991	-0.1041	0.17265	-0.266	6	0.07787
	7	-8.3803	0.02053	-5.4745	0.60848	0.00226	-5.2228	-0.3185	3.1031	0.37536	-0.1863	0.68164	7	2.1494
	8	-9.0783	0.00093	-5.0444	0.50532	-0.001	-4.7437	-0.4488	3.0964	0.29859	-0.1276	0.57327	8	-7.5481
	9	1.7608	0.24409	-0.0416	0.04756	0.03769	-6.5544	-0.0476	4.9861	0.07781	0.12572	-0.2168	9	-2.813
	10	-5.1906	-4.7786	0.02219	0.01032	0.00962	0.90305	-0.0206	-0.324	0.01262	0.00997	-0.0195	10	-2.0302
	11	-4.5874	-4.0573	-0.0163	-0.0195	-0.0071	-1.5501	-0.0299	1.3464	0.01805	0.02262	0.06356	11	4.4198
	12	4.7757	6.2878	0.03415	0.02663	-0.0033	0.83395	-0.041	-0.3522	0.02143	0.01544	-0.0382	12	0.84214
	13	-0.9222	-1.358	-0.0211	0.00344	0.02565	2.1847	-0.0145	-1.5593	0.01791	0.00414	-0.0271	13	-5.5393
	14	-3.1719	-2.7549	-0.0174	-0.0198	-0.0139	-1.6142	-0.021	1.1995	0.00984	0.01518	0.06665	14	-4.4442
	15	-1.6518	-0.3637	0.04369	-0.0487	-0.0376	6.3723	0.05059	-4.8884	-0.0814	-0.1198	0.19628	15	-2.9507

Table G.6: Network 4 Weights, Model C

Hidden Layer												Output Layer	
		Input Node i											Output
		0	1	2	3	4	5	6	7	8	9	10	0	0.7708
Hidden Node j	1	-2.86	-0.0162	-0.0363	-2.8986	0.01315	-0.0311	-0.0398	0.05517	-1.1936	-0.003	2.2032	1	5.3228
	2	2.843	-0.0126	0.0301	5.9324	-0.0586	-0.0107	0.00601	-0.0863	0.50464	-0.0035	-0.2156	2	-1.689
	3	-2.019	-0.0094	0.01388	-2.81	-0.0187	0.00137	0.00162	-0.0288	0.59485	0.0026	-0.4191	3	1.651
	4	-1.0464	0.10286	0.16671	-1.0266	-0.0209	-0.0275	0.55639	-0.2177	-4.3763	0.23473	-0.3879	4	0.0693
	5	-0.4672	0.08232	0.03978	-3.4364	-0.0917	-0.0421	0.06973	-0.0972	-4.441	-0.0089	3.1095	5	3.0285
	6	-0.7117	0.04136	0.04544	1.0925	-0.0566	0.03244	-0.0518	0.10927	-3.3861	-0.0428	2.7041	6	-0.993
	7	-1.1446	0.07768	-0.0034	-2.1601	-0.1628	-0.0555	0.04679	-0.1781	-5.0882	-0.0083	3.6825	7	1.1749
	8	-3.5077	0.02356	-0.0043	-3.9528	-0.0852	-0.0095	-0.004	0.06882	-3.58	-0.0452	2.9776	8	-6.7256
	9	-2.5909	-0.0049	0.01244	-2.6135	-0.0107	0.00406	0.00149	-0.025	0.8718	0.00445	-0.5594	9	-2.3579
	10	-0.3354	0.02492	0.03451	0.6692	-0.0648	0.02467	-0.0413	0.08913	-3.1541	-0.0508	2.4669	10	2.516
	11	-1.0539	0.06673	0.06715	1.5726	-0.0546	0.04983	-0.076	0.14123	-3.8444	-0.0352	3.135	11	1.3453
	12	0.97488	0.04509	-0.0328	-2.1706	0.44155	-0.0187	0.04363	-0.2237	-1.6173	0.18516	0.8049	12	-2.3244
	13	-1.0446	-0.0376	0.02635	2.1919	-0.3865	0.01641	-0.04	0.20733	1.6591	-0.1647	-0.8286	13	3.0251
	14	-2.4735	0.00045	-0.0075	-4.1176	0.0206	0.00065	0.00028	0.02657	-0.5437	0.00218	0.28224	14	-2.6641
	15	2.9646	-0.004	0.01974	2.9927	0.0105	0.0168	0.02102	-0.0727	1.6673	0.01594	-2.1121	15	-2.2245

Table G.7: Network 5 Weights, Model C

Hidden Layer												Output Layer	
		Input Node i											Output
Hidden Node j		0	1	2	3	4	5	6	7	8	9	10	0	-2.64
	1	2.3737	-1.7532	0.01314	0.00988	0.00344	0.84861	0.02648	-0.2962	0.00894	-0.0158	-0.0082	1	-4.0246
	2	8.5216	-1.0956	4.2612	-0.1161	0.01509	5.4086	0.03486	-3.6609	-0.1169	-0.0564	-0.2287	2	8.8856
	3	3.0691	2.4332	0.00375	0.00462	0.00816	1.1993	-0.0062	-1.0364	0.00872	0.00487	-0.0301	3	-1.7704
	4	-1.4695	0.8184	-0.0734	-0.0137	-0.0191	-1.7166	-0.067	0.17096	-0.0253	0.05024	0.02623	4	1.4196
	5	1.5987	0.05889	-0.0069	-0.0061	-0.0015	0.48933	-0.0191	-0.5381	0.01248	-0.0028	0.00453	5	4.2479
	6	0.08982	2.0131	-0.0054	-0.0041	-0.0045	-0.5809	0.00303	0.3873	-0.0076	-0.0017	0.00529	6	1.7266
	7	1.7673	-0.9408	0.03247	0.01347	0.00945	1.1386	0.04087	-0.255	0.00912	-0.02	-0.0197	7	6.1091
	8	2.3431	2.6709	0.03993	-0.0182	0.0209	4.2894	0.02313	-3.2562	-0.009	-0.0062	-0.023	8	0.35477
	9	0.05402	0.49973	0.03589	-0.0794	-0.0729	2.1131	0.01484	-2.1144	0.01899	-0.0746	-0.0565	9	-3.0495
	10	0.12659	-0.3924	-0.0265	0.05296	0.0517	-2.3677	-0.0031	2.1509	-0.0114	0.04646	0.08254	10	-7.0144
	11	-0.4297	0.21168	0.02026	-0.0356	-0.0377	2.6708	-0.0042	-2.281	0.00638	-0.0273	-0.1029	11	-3.7661
	12	-0.0289	-0.0027	-0.0539	0.03709	0.01895	-8.6013	-0.015	6.4408	0.048	0.04315	-0.0811	12	0.13826
	13	2.5482	3.2496	0.01143	0.01417	0.01073	-0.5328	0.01564	0.42904	-0.0104	0.00956	-0.0081	13	0.75872
	14	7.666	-1.4095	4.6467	-0.0968	0.06226	5.9635	0.1524	-3.9857	-0.2431	-0.0804	-0.1686	14	-5.2834
	15	-7.8153	1.663	-5.0478	0.07226	-0.1129	-6.4326	-0.2885	4.3274	0.3647	0.11193	0.08564	15	-2.1374

Table G.8: Network 6 Weights, Model C

Hidden Layer												Output Layer	
		Input Node i											Output
Hidden Node j		0	1	2	3	4	5	6	7	8	9	10	0	0.7708
	1	-1.3973	-0.0776	0.07096	0.58553	-0.0542	0.0098	0.04876	0.01686	-1.8351	0.01986	-0.3196	1	5.3228
	2	-2.965	0.00887	0.02809	-2.0087	-0.1346	0.00357	0.00169	0.01396	-3.0388	-0.0624	2.2498	2	-1.689
	3	1.7323	-0.0028	0.00083	2.5078	0.01696	-0.0119	0.01012	-0.0041	-0.4343	-0.0102	0.265	3	1.651
	4	1.2981	-0.0085	0.01454	-0.204	0.00791	0.03644	-0.0236	0.02792	5.3419	0.05894	-4.1552	4	0.0693
	5	0.2822	-0.071	0.06704	1.6649	-1.0898	0.14379	-0.0534	0.20407	1.1706	-0.2038	-0.7259	5	3.0285
	6	-1.1316	-0.0082	-0.0429	2.7713	0.00124	-0.0136	0.01964	-0.0548	-0.1804	-0.0267	-0.0272	6	-0.993
	7	0.72684	0.00581	0.01233	1.119	-0.0505	0.00928	-0.0231	0.05752	-1.9081	-0.0262	1.4583	7	1.1749
	8	-2.0789	0.00655	0.0164	-0.356	-0.0263	0.0287	-0.0086	0.09514	-2.5914	0.02083	1.8402	8	-6.7256
	9	3.3785	-0.0055	0.0024	2.9692	0.03408	0.014	-0.0053	0.01844	0.98655	0.02369	-0.7012	9	-2.3579
	10	1.7128	0.03728	-0.0405	-0.66	0.02656	0.00095	-0.0281	-0.0292	1.0905	-0.013	0.30132	10	2.516
	11	3.5267	-0.0149	-0.0291	2.3015	0.10408	-0.0133	0.00418	-0.0618	3.3825	0.04055	-2.5604	11	1.3453
	12	1.0657	-0.0078	-0.0002	-0.2966	0.04493	0.02934	-0.0249	0.05555	5.0155	0.06105	-3.8203	12	-2.3244
	13	-2.4919	-0.0013	0.01238	1.6668	0.01093	-0.0051	0.00597	0.00782	0.09916	0.01112	-0.2512	13	3.0251
	14	-0.1534	-0.0167	0.01115	1.1794	-0.341	0.05688	-0.0257	0.06994	1.3689	-0.0535	-0.982	14	-2.6641
	15	0.02228	-0.0438	0.0346	1.4351	-0.755	0.09998	-0.0401	0.13644	1.3066	-0.1444	-0.867	15	-2.2245

Table G.9: Network 7 Weights, Model C

Hidden Layer												Output Layer	
		Input Node i											Output
		0	1	2	3	4	5	6	7	8	9	10	0	-1.0421
Hidden Node j	1	0.29658	-0.0141	-0.0309	-0.0282	0.0318	0.12493	-0.108	0.70537	0.04478	-0.0072	0.38099	1	-2.294
	2	-0.807	0.01985	-0.7735	-0.0127	-0.0455	-0.9183	-0.0426	0.52013	-0.0848	0.022	0.13965	2	0.70764
	3	0.11891	-0.1814	0.00624	-0.0374	-0.0955	-0.1021	-0.3581	1.45	0.15731	0.18957	1.1537	3	0.43764
	4	0.00092	-0.0058	-0.0071	-0.0015	0.01035	0.58589	-0.2006	-0.9115	0.0279	-0.0011	-0.1888	4	1.913
	5	1.3834	-0.0497	-0.0589	-0.0415	-0.0605	-0.2327	-0.2265	0.75495	0.02834	0.08235	0.45604	5	1.3425
	6	-1.5417	0.00657	0.01809	0.01709	-0.0036	-5.4811	0.00239	4.0208	0.02236	0.01346	0.02319	6	1.7454
	7	2.4975	0.00342	-0.0485	0.0007	0.0257	0.46263	0.58071	0.8445	-0.0298	-0.0439	0.74549	7	2.6749
	8	2.4916	0.00343	-0.1016	0.01503	-0.0041	2.5278	0.06419	-1.725	-0.0143	0.01201	-0.066	8	-1.2668
	9	-1.5298	0.00821	-1.0743	-0.016	-0.0102	-0.1276	0.01382	0.09272	-0.0188	0.0073	0.02859	9	-2.831
	10	-0.8849	0.02084	-0.0547	-0.0154	0.02564	0.65024	0.26213	-0.0824	-0.0339	-0.0525	0.40157	10	2.0697
	11	2.7534	0.00023	-0.0663	0.00229	0.02421	0.17267	0.71599	1.4144	-0.0414	-0.0468	0.85804	11	-1.5702
	12	-0.4569	0.11339	-0.01	-0.0347	0.06058	0.27281	0.17989	-0.8778	-0.0786	-0.104	-0.8135	12	0.88128
	13	1.3115	-0.0046	-0.0421	-0.0139	0.0046	5.2061	0.00329	-3.7179	-0.0241	-0.0095	0.01193	13	1.7214
	14	1.2335	-0.0106	1.5069	0.03541	0.01028	-0.0586	-0.0125	-0.0031	0.00911	-0.0071	-0.0154	14	-1.104
	15	0.05341	0.01384	0.00568	-0.0184	-0.0176	-0.6062	0.06775	-0.1656	-0.0345	-0.0058	0.02125	15	-3.3747

Table G.10: Network 8 Weights, Model C

Hidden Layer												Output Layer	
		Input Node i											Output
		0	1	2	3	4	5	6	7	8	9	10	0	-1.0428
Hidden Node j	1	-1.2574	-0.0059	0.00273	-0.0228	0.03303	0.00759	0.01386	0.26062	0.9471	0.02138	-0.0044	1	2.4652
	2	0.66971	0.00067	0.00692	0.00479	0.00053	0.01149	-0.0358	-0.1895	-0.7155	-0.0223	0.90514	2	-1.5958
	3	0.58096	-0.0167	0.00089	-0.0166	0.54281	0.01157	-0.0499	-0.1035	0.53464	-0.1401	-0.3084	3	-1.4105
	4	-0.2155	0.00344	0.03141	0.02185	-0.2683	-0.0143	0.09566	0.30429	-0.4181	0.35422	0.36482	4	-0.9416
	5	-1.5312	-0.0241	0.0053	0.00185	-0.245	0.011	-0.0461	0.15813	-2.891	-0.0025	1.5341	5	0.86046
	6	-0.2191	-0.331	-0.0702	-0.6946	-0.292	0.02436	0.24744	-0.1647	-0.2198	0.0429	0.05694	6	-0.1174
	7	-0.9682	0.01077	-0.0092	-0.0148	0.06494	0.00674	0.00895	-0.0407	-5.0686	-0.0113	4.2035	7	2.0949
	8	-2.8242	0.01102	-0.0073	0.01908	-0.9963	-0.0087	-0.0273	0.07933	-0.2067	0.03248	0.1085	8	-1.5233
	9	-0.0773	0.18365	-0.1033	-0.0107	0.25432	0.12217	0.78405	-0.7428	0.65687	0.03805	-0.1622	9	-0.1257
	10	-2.5901	0.00393	-0.0501	-0.0018	0.00058	0.00226	0.04453	-1.8345	-0.2237	-0.0808	-1.0411	10	1.1096
	11	0.79832	-0.0069	0.00874	0.01707	-0.0682	-0.0076	-0.0085	0.06268	4.8668	0.01113	-3.9861	11	2.2371
	12	1.1438	0.00749	-0.0168	-0.0159	-0.0183	0.00412	0.0302	-0.0263	1.9525	-0.0002	-0.5146	12	1.7081
	13	1.6413	-0.0021	-0.0073	0.00954	-0.272	0.00695	-0.0318	-0.0089	1.5062	0.00256	-0.8013	13	-2.2865
	14	-2.4101	0.00581	-0.0362	0.00958	0.00206	-0.002	0.04226	-1.4554	-0.4857	-0.063	-0.7179	14	-1.7869
	15	-0.0783	-0.0096	0.00914	0.00329	-0.0381	0.00852	0.0114	-0.532	0.46762	0.04816	-0.5812	15	1.8833

Table G.11: Network 9 Weights, Model C

Hidden Layer

Hidden Node j	Input Node i 0	1	2	3	4	5	6	7	8	9	10
1	0.10341	0.01157	-0.0057	0.00845	0.15928	0.01716	-0.0502	0.45347	-0.013	0.01018	0.15572
2	0.72549	-0.0092	-0.0713	-0.0043	0.03648	-0.0551	-0.0332	-0.1604	-0.0036	1.4241	-2.5773
3	0.50059	-0.0022	0.33083	0.0085	-0.3092	0.20365	0.04943	0.90098	-0.0582	-1.0517	2.8189
4	-0.2346	-0.3267	1.944	-0.1027	-0.393	0.06168	0.84626	-1.4963	0.17761	-0.5972	-0.2835
5	-1.539	-0.0412	0.0023	0.0134	0.10386	0.0026	0.13419	0.29107	0.10365	-1.8266	-0.0464
6	-0.7338	-0.0121	0.03527	0.00882	-0.0038	-0.0387	0.10532	0.21223	0.06346	0.10864	-2.1225
7	2.3551	0.05756	0.03165	-0.0091	-0.1371	0.04193	-0.1387	-0.3861	-0.1093	1.848	1.276
8	5.9072	-0.0176	0.11273	0.03153	0.01319	0.02252	-0.0277	2.706	0.03054	-0.0296	1.638
9	-1.0966	-0.0263	0.01156	0.01753	0.10225	0.01239	0.11176	0.24901	0.11664	-1.9943	1.2399
10	-0.3508	-0.009	-0.2463	-0.0058	0.2488	-0.1685	-0.0242	-0.729	0.0937	1.1038	-2.5803
11	0.01396	-0.0333	-0.0379	-0.0228	-0.6536	-0.0869	0.21648	-0.2151	0.03502	0.06575	-0.1445
12	-5.443	0.02929	-0.2123	-0.0411	-0.0484	-0.0231	0.03225	-3.1368	-0.0699	-0.0222	-1.9416
13	6.6495	-0.0717	-0.0642	-0.0989	0.18606	0.01725	4.642	-0.7922	-0.0118	-0.0254	-0.0982
14	-6.0071	0.04777	-0.1208	0.01942	0.12824	0.01788	-4.2786	-0.2905	-0.0374	0.03679	-0.6017
15	5.6526	-0.0652	0.13647	-0.0528	-0.0784	-0.0178	4.3314	0.25391	0.04247	-0.0186	0.61172

Output Layer

Hidden Node j	Output
0	0.7708
1	5.3228
2	-1.689
3	1.651
4	0.0693
5	3.0285
6	-0.993
7	1.1749
8	-6.7256
9	-2.3579
10	2.516
11	1.3453
12	-2.3244
13	3.0251
14	-2.6641
15	-2.2245

G.1 Batch Parameter Analysis, Model C

Table G.12: Results for Example Model, Products 1 and 2, Model C

		Product 1						Product 2					
		Delay		Fill Rate		FGI		Delay		Fill Rate		FGI	
r_3	r_4	NN	Sim	NN	Sim	NN	Sim	NN	Sim	NN	Sim	NN	Sim
1	1	22.0	22.8	0.842	0.831	1.9	1.9	6.1	7.7	0.929	0.918	2.8	2.8
1	2	21.2	21.8	0.840	0.835	1.9	1.9	7.1	7.8	0.921	0.912	2.8	2.7
1	3	21.0	21.1	0.838	0.833	1.9	1.9	9.1	10.2	0.886	0.890	2.5	2.5
1	4	21.0	22.8	0.836	0.831	1.9	1.9	14.8	17.9	0.802	0.815	2.0	2.0
2	1	22.4	20.3	0.844	0.840	1.9	1.9	6.3	6.2	0.926	0.927	2.8	2.8
2	2	21.5	20.1	0.840	0.842	1.9	1.9	7.5	6.7	0.917	0.918	2.7	2.7
2	3	21.3	21.1	0.837	0.838	1.9	1.9	10.0	10.5	0.881	0.881	2.5	2.5
2	4	21.7	24.4	0.834	0.822	1.8	1.8	16.1	18.1	0.797	0.810	2.0	2.0
3	1	27.3	26.5	0.798	0.807	1.7	1.7	6.5	6.7	0.924	0.922	2.8	2.8
3	2	26.4	25.9	0.794	0.804	1.7	1.7	8.0	8.3	0.914	0.913	2.7	2.7
3	3	26.6	27.2	0.791	0.796	1.7	1.7	10.8	10.3	0.876	0.884	2.5	2.5
3	4	27.8	29.3	0.787	0.791	1.6	1.7	17.5	17.4	0.792	0.815	2.0	2.0
4	1	44.2	41.2	0.685	0.703	1.3	1.3	6.7	7.3	0.922	0.917	2.8	2.7
4	2	43.8	48.5	0.682	0.679	1.3	1.3	8.5	8.5	0.910	0.908	2.7	2.7
4	3	44.9	45.9	0.678	0.685	1.3	1.3	11.7	12.1	0.871	0.872	2.5	2.5
4	4	47.1	44.7	0.674	0.693	1.3	1.3	18.9	19.8	0.787	0.803	2.0	2.0

Table G.13: Results for Example Model, WIP, Model C

		WIP					
		WIP3		WIP4		WIP5	
r_3	r_4	NN	Sim	NN	Sim	NN	Sim
1	1	2.1	2.1	1.9	1.9	0.6	0.6
1	2	2.1	2.1	1.3	1.4	0.6	0.6
1	3	2.1	2.1	1.0	1.0	0.7	0.7
1	4	2.1	2.1	0.9	0.9	0.8	0.9
2	1	1.5	1.5	1.9	1.9	0.6	0.6
2	2	1.6	1.5	1.4	1.4	0.7	0.7
2	3	1.6	1.6	1.0	1.0	0.8	0.8
2	4	1.6	1.6	0.9	0.9	0.9	0.9
3	1	1.2	1.2	2.0	2.0	0.6	0.6
3	2	1.2	1.1	1.4	1.4	0.7	0.7
3	3	1.2	1.2	1.1	1.0	0.9	0.9
3	4	1.2	1.2	0.9	0.9	1.0	1.0
4	1	1.1	1.1	2.0	2.0	0.7	0.7
4	2	1.1	1.1	1.5	1.5	0.8	0.8
4	3	1.1	1.1	1.1	1.1	0.9	1.0
4	4	1.1	1.1	0.9	1.0	1.1	1.1

G.2 Simulated Annealing Results, Model C

Table G.14: Simulated Annealing Algorithm Parameters

Number of Cooling Cycles	20
Initial Temperature	1000
Number of Iterations at First Temperature	5000
Number of Iterations at Subsequent Temperatures	2000

Table G.15: Initial Points for Simulated Annealing Experiments

Exp. No.	Initial Point										Initial Cost
	z_1	k_1	z_2	k_2	z_3	k_3	r_3	z_4	k_4	r_4	
1	3	3	3	3	3	3	3	3	3	3	$ 137.79
2	4	4	2	10	3	6	1	3	3	1	$ 171.75
3	5	6	0	5	9	6	2	1	2	2	$ 336.76
4	8	6	3	4	9	10	5	1	3	1	$ 178.60
5	4	2	8	8	1	3	1	5	3	1	$ 143.58
6	6	3	2	8	7	4	1	4	2	2	$ 196.59
7	3	6	4	8	3	7	3	6	3	3	$ 139.88
8	8	3	3	3	10	6	5	5	3	1	$ 191.90
9	8	4	4	7	6	10	1	6	3	1	$ 175.50
10	6	6	2	3	1	2	1	7	2	2	$ 181.54
11	8	6	10	9	9	3	1	8	2	2	$ 238.73
12	6	8	10	8	6	7	1	10	4	4	$ 213.34
13	10	4	9	6	7	9	5	10	2	2	$ 239.69
14	0	6	2	5	7	3	3	3	6	3	$ 378.13
15	7	6	10	8	5	7	7	1	6	6	$ 138.00
16	5	6	3	4	6	1	1	5	6	4	$ 151.91
17	8	9	3	9	4	9	2	5	6	1	$ 169.44
18	2	2	0	8	8	4	3	10	5	3	$ 413.44
19	5	9	1	3	1	9	6	8	6	5	$ 320.11
20	8	7	2	10	2	1	1	3	10	7	$ 289.59
21	9	5	7	6	1	6	6	3	8	2	$ 135.22
22	2	5	8	5	6	1	1	6	9	7	$ 205.54
23	2	7	7	8	3	8	5	6	9	4	$ 229.28
24	5	3	5	4	4	5	5	9	10	8	$ 135.26
25	2	9	3	3	8	8	6	7	10	4	$ 206.18

Table G.16: Results of Simulated Annealing Experiments

Exp.	Final Solution										Final
No.	z_1	k_1	z_2	k_2	z_3	k_3	r_3	z_4	k_4	r_4	Cost
1	5	2	5	3	1	3	1	1	4	1	$ 103.52
2	5	2	5	3	1	3	1	1	4	1	$ 103.52
3	5	2	5	3	1	3	1	1	4	1	$ 103.52
4	5	2	5	3	1	3	1	1	4	1	$ 103.52
5	5	2	5	3	1	3	1	1	4	1	$ 103.52
6	5	2	5	3	1	3	1	1	4	1	$ 103.52
7	5	2	5	3	1	3	1	1	4	1	$ 103.52
8	5	2	5	3	1	3	1	1	4	1	$ 103.52
9	5	2	5	3	1	3	1	1	4	1	$ 103.52
10	5	2	5	3	1	3	1	1	4	1	$ 103.52
11	5	2	5	3	1	3	1	1	4	1	$ 103.52
12	5	2	5	3	1	3	1	1	4	1	$ 103.52
13	5	2	5	3	1	3	1	1	4	1	$ 103.52
14	5	2	5	3	1	3	1	1	4	1	$ 103.52
15	5	2	5	3	1	3	1	1	4	1	$ 103.52
16	5	2	5	7	1	3	1	1	2	1	$ 103.91
17	5	2	5	3	1	3	1	1	4	1	$ 103.52
18	5	2	5	3	1	3	1	1	4	1	$ 103.52
19	5	2	5	3	1	3	1	1	4	1	$ 103.52
20	5	2	5	3	1	3	1	1	4	1	$ 103.52
21	5	2	5	3	1	3	1	1	4	1	$ 103.52
22	5	2	5	3	1	3	1	1	4	1	$ 103.52
23	5	2	5	3	1	3	1	1	4	1	$ 103.52
24	5	2	5	3	1	3	1	1	4	1	$ 103.52
25	5	2	5	3	1	3	1	1	4	1	$ 103.52

Appendix H

Results for Model D

Table H.1: Parameter Ranges for Model D

	Low	Mid	High
z_1	0, 3	4, 6	7,10
k_1	3, 5	6, 8	9,12
z_2	0, 3	4, 8	9,12
k_2	1, 3	4, 6	7,10
z_3	0, 3	4, 6	7,10
k_3	3, 6	7,10	11,15
z_4	0, 5	6,10	11,16
k_4	2, 4	5, 7	8,10

Table H.2: Networks for Model D

Network	Description	Hidden Nodes
1	Average customer delay time	15
2	Average fill rate	15
3	Average finished goods inventory	15
4	WIP Product 2	15
5	WIP Product 3	15
6	WIP Product 4	15
7	WIP Product 5	18
8	WIP Product 6	15
9	WIP Product 7	18
10	WIP Product 8	15

Table H.3: Network 1 Weights, Model D

		Hidden Layer: Input Node i								
		0	1	2	3	4	5	6	7	8
Hidden Node j	1	9.6399	0.46097	4.6809	-0.55398	-0.35708	1.879	0.8949	1.7878	0.77406
	2	2.8242	0.80161	0.01588	-0.12215	0.0066	1.1112	0.00638	-0.00567	-0.00429
	3	-10.466	-0.41388	-4.7984	0.43375	0.33653	-1.7907	-0.83379	-1.538	-0.61763
	4	2.6851	1.0661	0.04918	2.6963	-0.02444	-0.6939	-0.01835	-0.00747	0.00461
	5	-0.30763	-1.0185	-0.12001	1.2762	0.04134	0.26065	0.01757	-0.05042	-0.00323
	6	0.08551	1.2516	0.28845	-1.2622	-0.05162	-0.74942	0.07814	-0.06535	-0.03707
	7	0.54102	0.74916	0.08725	-1.4622	-0.04557	-0.27463	-0.0471	0.09196	0.01178
	8	1.9199	1.0005	0.01648	-0.11406	0.00984	1.4576	-0.00045	-0.00405	-0.00097
	9	8.3323	1.7562	0.18428	0.57243	0.09	1.6957	0.51426	6.0494	-0.03995
	10	3.0313	1.7566	0.54146	1.6841	-0.131	1.7514	-0.11092	0.40034	-0.11494
	11	-0.06076	-1.286	-0.22486	1.1823	0.04555	0.5401	-0.04411	0.03056	0.01981
	12	-8.2087	-1.7343	-0.18536	-0.49486	-0.08899	-1.6974	-0.47445	-5.8045	0.02361
	13	-2.9818	-1.7561	-0.4486	-1.6868	0.12951	-1.6227	0.12212	-0.38147	0.08676
	14	4.2944	3.0957	0.01955	0.15606	0.00489	-0.14327	0.01093	-0.02511	0.00059
	15	2.5529	1.1779	0.05609	2.679	-0.02984	-0.60525	-0.02109	-0.00299	0.0085

		Output Layer: Output
	0	0.64983
Hidden Node j	1	2.0591
	2	-8.2602
	3	5.5831
	4	-2.7571
	5	-4.9791
	6	2.2564
	7	-2.631
	8	2.7622
	9	3.3051
	10	2.0271
	11	4.4727
	12	3.8757
	13	2.3966
	14	-3.2819
	15	2.8189

Table H.4: Network 2 Weights, Model D

		Hidden Layer: Input Node i								
		0	1	2	3	4	5	6	7	8
Hidden Node j	1	1.0137	0.23346	0.14826	-0.05942	-0.08926	1.8059	0.11374	2.5379	0.16008
	2	-0.92396	-1.7388	-0.00795	-0.3525	0.00266	0.99236	0.00655	0.0292	0.00148
	3	0.66981	2.015	0.00773	0.14625	-0.0022	-0.69164	-0.00413	-0.0132	0.00425
	4	-2.6912	-0.24082	-0.0942	-2.3673	0.00997	0.59396	0.01129	0.03913	-0.02909
	5	-2.3631	-3.3957	0.01219	-0.15765	0.00163	-0.60207	0.00092	-0.0387	0.00118
	6	-1.6343	-1.5759	0.01684	0.06367	-0.01014	-0.30536	0.00401	0.09513	-0.00229
	7	-2.1981	-0.53237	-0.06813	-1.7799	0.01364	0.43661	0.01109	0.02589	-0.01953
	8	1.8933	1.7203	-0.03632	-0.04745	0.04446	0.1761	-0.02739	-0.66727	-0.01149
	9	1.1215	-0.33788	0.04224	0.49472	-0.01578	1.3513	-0.00811	0.13804	0.02179
	10	4.1271	3.4781	0.0373	1.9914	-0.01156	-0.47651	-0.01803	-0.01942	0.03156
	11	0.43185	1.814	0.00907	0.26184	-0.00602	0.13749	-0.00261	0.01202	0.0109
	12	-0.81982	-0.0743	-0.05221	-0.19921	0.01276	-1.9614	0.0216	-0.00653	-0.03013
	13	-3.3762	-8.5174	-0.01418	0.15989	0.13181	0.15727	0.0423	0.3381	0.08936
	14	-5.7707	-5.2657	0.00661	0.07492	0.01373	-2.2948	-0.01517	-0.19851	-0.03895
	15	-1.9294	-1.363	0.02273	0.74564	-0.01573	-0.36602	-0.00499	-0.03963	-0.0071

		Output Layer: Output
	0	-5.5172
Hidden Node j	1	-0.1717
	2	-4.9155
	3	-6.4713
	4	-3.1025
	5	-4.3953
	6	16.749
	7	5.8959
	8	2.908
	9	1.6181
	10	1.7513
	11	7.0275
	12	1.0383
	13	0.1792
	14	-0.96659
	15	-4.9894

Table H.5: Network 3 Weights, Model D

Hidden Layer										Output Layer	
		Input Node i									Output
		0	1	2	3	4	5	6	7	8	0	-3.4093
Hidden Node j	1	-3.9074	-3.0241	-0.00356	-0.37273	-0.00367	-0.34546	-0.01044	-0.05137	0.00654	1	1.5613
	2	-0.71741	1.2211	-0.01203	0.18165	0.00492	-0.1449	-0.00571	-0.02367	-0.00087	2	5.2159
	3	-2.8061	1.3298	0.00904	1.1671	0.00121	-0.51998	0.00435	-0.00186	-0.0064	3	2.4451
	4	4.1261	-0.66389	0.27299	-0.02896	0.02991	1.2095	0.2504	2.4489	-0.00474	4	-1.3869
	5	-1.781	0.66397	-0.00253	0.54709	-0.00093	-0.79339	0.00082	0.01257	-0.00114	5	-6.4025
	6	-0.87569	0.64443	-0.00943	0.20419	-0.00127	-1.2807	0.00446	0.08264	0.00775	6	2.3155
	7	-1.2882	-0.86624	-0.01288	-0.78185	-0.00152	-0.2275	-0.00574	-0.0192	0.00972	7	-2.4479
	8	-0.40225	0.7911	-0.01165	-0.94453	0.00077	0.45898	0.00661	0.03264	0.00261	8	2.7937
	9	-0.76919	1.7243	-0.02621	0.02979	0.03057	-0.75402	0.00384	-0.42998	-0.01309	9	-2.5891
	10	-0.63858	1.5699	-0.01018	0.0657	0.04516	-1.0121	0.01072	-0.74615	-0.02693	10	1.2806
	11	4.7409	-0.59865	0.2168	-0.07706	0.01318	1.104	0.21542	2.3304	0.00273	11	2.9429
	12	-0.1321	0.06233	0.51198	-0.97832	-0.76775	0.8953	-0.22023	0.85351	0.24202	12	0.02038
	13	-1.1218	2.1531	0.00638	-0.21891	0.04485	-0.36613	-0.00708	0.99103	0.07478	13	-0.2114
	14	2.1656	-0.31832	0.01372	1.6961	0.01725	1.3644	-0.01164	-0.03129	-0.00141	14	0.43562
	15	-0.84495	1.1534	-0.01247	-0.61126	-0.00267	0.86771	0.00484	0.05272	-0.003	15	-1.8196

Table H.6: Network 4 Weights, Model D

Hidden Layer										Output Layer	
		Input Node i									Output
		0	1	2	3	4	5	6	7	8	0	-1.6457
Hidden Node j	1	-1.6944	-0.01225	-0.09017	-0.42012	-0.00815	1.0559	0.0941	0.36226	0.00534	1	-4.6259
	2	-2.7	-0.04471	0.30122	-0.14329	0.10248	-0.46479	-2.3793	-2.279	-0.13823	2	-0.19544
	3	1.8897	0.00329	-0.00345	4.629	0.02697	-2.2185	0.03105	-0.2192	0.05721	3	-1.3213
	4	1.7554	0.00284	-0.01949	4.6255	0.02329	-2.0046	0.0274	-0.19767	0.0456	4	1.4786
	5	-0.79819	0.00951	0.11691	0.45122	0.01088	-1.765	0.02545	0.28225	-0.0051	5	-1.6197
	6	7.9983	0.0515	-0.04003	-0.42992	0.14989	10.788	0.06687	0.45776	-0.04204	6	-0.11532
	7	4.202	-0.00761	2.1773	0.13529	-0.00322	0.52926	0.04258	0.1383	0.0226	7	1.1679
	8	-1.8277	0.01039	0.09982	0.18615	0.00521	-0.8676	-0.10845	-1.1001	-0.01272	8	5.2853
	9	-1.6728	-0.01585	-0.12069	-0.16589	-0.0099	1.4131	0.00893	-0.1268	-0.00291	9	3.4654
	10	-2.1149	-0.00226	-1.7267	0.07998	0.03065	-1.744	-0.09969	-0.43441	-0.08931	10	0.20427
	11	1.8779	-0.01862	-0.1062	-0.16147	0.00749	1.0726	0.02827	2.1003	0.02356	11	1.7092
	12	0.72598	-0.01397	0.09489	-0.07032	0.03228	-0.03477	-1.1461	-0.72334	-0.08227	12	-0.61564
	13	0.60117	0.00261	0.00399	0.00654	-0.00785	-0.25159	0.07205	-1.0091	-0.00396	13	-2.3628
	14	3.2381	0.05487	-0.01934	5.0637	-0.00089	1.0016	0.03181	-0.00805	0.02357	14	-0.19524
	15	-1.0975	-0.00019	0.04286	1.0415	0.00119	-0.86551	-0.03834	-0.04213	-0.01162	15	2.7943

Table H.7: Network 5 Weights, Model D

Hidden Layer										Output Layer	
		Input Node i									Output
		0	1	2	3	4	5	6	7	8	0	-1.3663
Hidden Node j	1	-1.3177	-0.01089	-0.37696	-0.0546	-0.00459	0.41395	0.38069	0.17951	0.01574	1	-4.2974
	2	1.9063	0.00773	0.5942	0.04975	0.001	0.01144	0.16965	0.10331	0.01701	2	4.6186
	3	-1.7346	0.01149	0.0161	0.07534	-0.02295	-0.76861	-0.00565	-2.9444	-0.00273	3	-1.5198
	4	-1.9132	-0.00754	0.00595	-2.481	0.00757	-1.3575	-0.00303	-0.07781	-0.0198	4	0.49517
	5	-1.7196	0.00881	-0.00132	0.04248	-0.03	-0.56106	-0.0097	-3.6469	-0.00162	5	0.92024
	6	0.00308	-0.00428	0.00572	-0.06918	0.01789	0.50664	-0.03017	-1.031	-0.00495	6	2.1211
	7	-0.93752	0.02842	-1.3928	-0.04816	-0.02	-0.47298	0.1804	0.12694	0.03137	7	0.69545
	8	1.0734	-0.01036	-0.04881	-0.68269	-0.00263	4.6124	-0.00785	0.47431	0.02486	8	0.25987
	9	-0.60976	-0.01338	-0.02961	1.8908	0.01267	-0.76909	-0.00688	-0.0877	-0.02055	9	0.80235
	10	-0.36844	0.11791	-0.15653	0.02102	-1.5076	0.92081	0.01697	-0.71049	-0.4742	10	0.02258
	11	-1.2961	-0.00327	0.17477	0.02111	0.01535	0.38712	-0.2844	-0.37001	-0.01933	11	-6.539
	12	2.0663	0.0113	-0.11256	8.3584	-2.7E-05	-0.95607	0.00187	-0.26287	0.02994	12	0.09245
	13	-1.3031	-0.00377	0.16444	0.03949	0.03339	0.2241	-1.0154	-0.84312	-0.07882	13	1.225
	14	-1.1664	-0.04121	-0.64285	-0.16645	-0.00406	0.64224	0.16188	0.20809	-0.01938	14	2.2492
	15	1.0591	-0.00466	-0.04803	0.26155	-0.00756	-1.2611	-0.0052	-0.21146	-0.00929	15	-4.3818

Table H.8: Network 6 Weights, Model D

Hidden Layer										Output Layer	
		Input Node i									Output
		0	1	2	3	4	5	6	7	8	0	0.04379
Hidden Node j	1	1.6009	-0.56003	-0.02581	-1.1168	-0.97122	-0.59072	-0.72598	-1.0514	0.1451	1	-0.06817
	2	0.74525	0.00303	-0.00957	-0.00304	0.00356	-0.00358	0.14585	-1.226	-0.00766	2	-3.4971
	3	2.0008	0.04856	1.5526	-0.04582	-0.05077	0.87279	-0.06813	0.15854	-0.00741	3	-1.2206
	4	-0.12887	0.20387	0.06665	0.22194	-0.27877	-0.21027	0.12559	-0.03079	0.04087	4	-0.54128
	5	2.115	0.05992	1.1604	0.00087	-0.04698	0.61327	0.36052	-0.01133	-0.00202	5	2.2987
	6	4.5393	-0.0582	0.03224	0.09816	-0.00729	0.00973	-0.4612	9.3467	0.0107	6	0.0936
	7	2.6711	0.00314	0.33331	0.00704	0.02823	0.23061	-1.0696	-2.059	0.01181	7	-0.71232
	8	0.69692	-0.05035	-0.43893	0.0343	0.00301	-0.28469	0.70704	0.21968	0.00882	8	1.7906
	9	-0.64155	0.04597	0.05407	-0.02139	-0.03023	0.03054	0.10758	-5.3111	0.03416	9	-0.17497
	10	-0.10155	0.0245	-0.23111	-0.04204	-0.05165	-0.19352	1.3775	1.9452	0.00674	10	0.33762
	11	0.18533	0.03181	-0.51288	-0.03001	-0.01806	-0.19541	0.34755	-0.34828	-0.02054	11	1.5901
	12	-2.3413	0.73129	-0.65122	1.8485	1.43	0.71118	0.55669	0.58432	-0.26074	12	-0.03732
	13	-0.69221	0.07435	0.51082	0.03964	-0.05353	0.191	0.44511	-0.25233	0.01346	13	2.0747
	14	-0.76113	0.0174	-0.1875	0.00187	-0.00102	-0.11686	1.0613	0.2569	0.00156	14	-2.605
	15	-1.5459	0.04084	0.2711	0.00912	-0.00282	0.13055	-2.6865	-0.25066	-0.00441	15	0.35386

Table H.9: Network 7 Weights, Model D

Hidden Layer										Output Layer	
		Input Node i									Output
		0	1	2	3	4	5	6	7	8	0	0.8038
Hidden Node j	1	0.18681	-0.185	1.4985	0.29792	0.5165	2.2903	-1.0448	-1.7883	0.12096	1	0.07943
	2	-2.4587	0.03557	0.13208	0.10291	-0.01025	-1.1622	0.13606	-2.4217	-0.0534	2	-2.8782
	3	0.74912	0.04968	0.25731	-1.2138	-0.00963	0.78011	0.02966	0.04962	5.8E-05	3	-2.1515
	4	2.3905	-0.09331	-0.03142	-0.21109	-0.01535	4.9857	0.01847	0.17634	-0.00625	4	2.2616
	5	-6.7474	0.02277	0.25414	0.00241	-0.07084	-0.2821	-2.6132	-2.9008	0.01577	5	5.4925
	6	6.1595	-0.01885	-0.25508	-0.02404	0.10202	0.33399	2.9552	3.2286	-0.04266	6	2.0353
	7	-0.65702	0.11077	1.2242	-0.16253	0.09292	0.57234	0.10965	0.3394	0.03712	7	0.92671
	8	-0.49858	0.01916	0.72648	1.3033	-0.00633	-1.2748	-0.0229	0.06926	-0.01585	8	1.881
	9	2.7489	-0.02545	-0.09702	-0.06479	0.00429	0.91571	-0.11113	2.0953	0.03854	9	-4.662
	10	1.9666	-0.00918	-0.02463	5.6246	0.02972	-2.5592	0.05715	-0.34583	0.0359	10	1.9364
	11	2.3904	-0.08945	0.11671	-0.14449	-0.00588	4.2529	0.02936	0.18405	-0.00017	11	-3.1099
	12	-2.1049	0.00474	0.0145	-5.6217	-0.0376	2.8338	-0.07143	0.40053	-0.0497	12	1.7143
	13	-4.0015	-0.11645	-0.04157	-5.9213	-0.00283	-0.98898	-0.07406	0.02815	0.00445	13	0.36423
	14	-0.23167	-0.01171	-0.24993	-0.6934	-0.02673	-0.10023	0.00381	0.02862	-0.00377	14	4.1245
	15	2.3701	0.19727	-1.5452	0.76632	-0.08743	3.9843	-0.20943	0.37373	-0.04519	15	-0.30555
	16	-0.0455	0.01935	0.59885	1.06	-0.0395	-1.9081	-0.07765	-0.47336	-0.01909	16	-1.2046
	17	-0.21644	-0.04487	0.49956	1.0526	0.04715	-1.4936	0.02397	0.62354	0.00965	17	-1.1389
	18	-3.8631	-0.02881	-2.5554	-0.19106	-0.02696	-0.45731	-0.03963	-0.15228	-0.00717	18	-2.3524

Table H.10: Network 8 Weights, Model D

Hidden Layer										Output Layer	
		Input Node i									Output
		0	1	2	3	4	5	6	7	8	0	-0.58852
Hidden Node j	1	0.74362	0.04872	0.69158	0.46764	-0.04143	-0.5032	-0.07048	-0.20194	0.00441	1	4.6568
	2	-2.7255	0.01016	-1.7826	-0.42137	0.80936	-0.05583	0.01958	0.0024	0.01071	2	-6.4082
	3	1.2806	0.1353	0.58894	0.03219	-1.098	0.21807	-0.09395	2.2804	-0.32192	3	1.0193
	4	-0.28859	-0.03092	-1.1931	-0.49152	0.23802	0.50341	0.07952	0.17507	0.01894	4	5.1902
	5	0.42164	-0.02714	0.8332	-0.65224	-0.31583	-0.16737	-0.02762	-0.07229	-0.01216	5	-4.1806
	6	-2.1644	0.01863	-2.1238	-0.20211	-0.10987	-0.60879	0.03853	-0.2254	-0.03176	6	-3.4063
	7	-3.3533	0.00507	-0.06288	0.00431	-2.58	-0.00446	0.00059	0.00448	0.01126	7	-3.7636
	8	-0.27359	-0.01649	0.89943	0.00389	1.2438	0.17556	-0.03675	0.05252	-0.06381	8	1.0591
	9	-0.62369	0.02553	-0.42927	1.0161	0.21347	0.08216	0.00027	0.04196	-0.01721	9	-2.2993
	10	-1.9729	0.02211	-2.0574	-0.13371	0.21127	-0.65498	0.01347	-0.2008	-0.03728	10	3.9158
	11	1.2555	0.11283	0.61615	-0.00152	-1.1706	0.31948	-0.06353	2.424	-0.31682	11	-0.95727
	12	6.5863	0.44713	1.4367	-0.61763	-1.7182	4.7091	-0.00137	-0.1299	0.39214	12	0.17589
	13	-2.0276	0.69182	1.7774	-1.0187	1.2076	0.20997	-2.0382	-2.371	-1.0306	13	-0.0298
	14	0.00408	-0.03019	1.5963	0.00303	-0.52341	-0.39924	-0.06673	-0.12559	-0.03923	14	2.6039
	15	2.9054	0.03201	2.1915	1.099	-0.66576	-0.26612	-0.08685	-0.07189	-0.01776	15	-2.781

Table H.11: Network 9 Weights, Model D

Hidden Layer										Output Layer	
		Input Node i									Output
Hidden Node j	0	1	2	3	4	5	6	7	8	0	-3.416
1	-1.1851	0.01952	-0.17334	-0.11661	-0.00236	0.12665	0.32124	-1.3109	0.03275	1	-4.1606
2	-2.7873	4.5499	6.2787	-2.7571	0.12124	8.3741	-4.7739	0.81281	1.0497	2	-0.02642
3	-0.71049	0.03845	-0.97381	0.06209	0.01518	-0.16776	0.45235	0.05686	0.00582	3	6.7148
4	0.20532	-0.29259	0.19513	-0.24706	0.10305	-0.44163	-2.6417	0.79473	-0.16767	4	-0.35805
5	-1.2633	0.0194	-0.97319	-0.02707	-0.01171	-0.16064	-0.24377	0.02639	0.02679	5	-5.1067
6	7.5615	-0.04959	0.03989	0.03471	0.02338	0.20098	-0.08257	8.8697	0.14886	6	-0.52013
7	2.1623	0.00314	-0.03988	-0.06549	0.00471	-0.09775	-0.81968	1.1284	-0.03065	7	-5.9497
8	0.55882	0.99222	-1.4699	-0.82126	0.34623	0.43985	-1.9852	-2.1932	-1.6142	8	1.0701
9	3.9113	0.06212	1.9341	-0.07829	0.06057	1.4731	-0.70623	0.14002	-0.00763	9	5.9161
10	-3.4414	-0.1241	-2.5524	0.13053	-0.10243	-1.7988	0.82253	-0.17047	-0.00195	10	2.556
11	1.6417	0.00891	-0.07396	-0.07321	0.01452	0.04986	0.75773	1.2819	0.02007	11	-3.8523
12	-1.3305	0.03159	0.17502	0.5005	-0.0167	-0.16346	0.29252	-0.86351	-0.04842	12	-2.4184
13	1.1924	0.02229	0.98792	0.14153	0.01524	-0.01343	-0.88218	-0.18576	-0.02144	13	8.0341
14	1.0036	0.07174	1.3006	0.43996	0.04818	-0.13209	-0.73569	-0.09472	-0.06267	14	-3.5048
15	1.8664	0.68779	-0.89766	-0.55119	0.41314	0.07336	0.08613	-2.6163	-0.26905	15	-0.2091
16	3.8462	0.04288	-0.20744	-0.00606	-0.00011	0.01785	2.6888	0.3433	0.03945	16	3.3589
17	6.5033	0.51525	6.2972	0.11085	-0.02686	2.7138	0.49445	0.83745	0.37127	17	0.1502
18	0.51592	1.0745	-1.5711	-0.84807	0.38892	0.50542	-2.23	-2.3256	-1.8067	18	-0.94026

Table H.12: Network 10 Weights, Model D

Hidden Layer										Output Layer	
		Input Node i									Output
Hidden Node j	0	1	2	3	4	5	6	7	8	0	-3.4404
1	-2.2933	0.01771	0.16032	0.08088	0.03933	0.03875	-3.0401	-0.02241	1.1923	1	1.5818
2	-2.527	0.04088	-2.0699	0.14209	-0.03938	-1.6047	0.5189	-0.11776	0.59263	2	1.1031
3	8.1405	0.03854	-0.05576	-0.02412	0.00711	-0.03989	3.216	3.6329	-0.57329	3	5.5097
4	-0.43003	-0.01108	0.01499	0.04036	0.02437	-0.01965	-1.0007	-0.03713	0.90464	4	-4.7241
5	-3.24	-0.00899	-0.08198	0.00619	0.01379	-0.0185	-0.15562	-0.01484	-2.0431	5	-2.4683
6	0.11751	-0.03541	0.24978	0.07854	0.03801	-0.05502	-2.1126	-0.05979	1.4872	6	1.1273
7	2.3969	0.00269	1.5068	0.08362	0.01216	0.50616	-0.588	0.03155	-0.39097	7	4.6313
8	-0.25581	0.00964	-0.13446	-0.01238	0.00225	-0.01706	-0.20609	0.00978	-0.53238	8	-6.6274
9	-0.26452	-0.01505	0.40263	-0.15402	0.00403	0.68948	-0.02405	0.07224	-0.13753	9	-2.9321
10	0.74481	-0.02459	-0.49035	-0.19444	0.02218	0.46025	0.22898	0.04851	0.09637	10	3.3238
11	-2.6646	-0.00296	-0.59036	-0.06314	0.01411	-0.03787	-1.3541	0.017	0.75899	11	-4.2902
12	6.2275	0.06468	-0.05185	-0.03547	-0.00485	-0.03141	2.682	3.0143	0.23062	12	-4.9548
13	-1.7795	-0.04279	-1.7804	-0.27352	0.01739	0.04237	0.20989	0.01631	0.5538	13	2.1622
14	0.50978	0.03202	0.44291	0.05644	0.00304	0.06959	-1.2949	0.00975	0.23911	14	2.9296
15	-5.7391	-0.06813	0.04926	0.04087	0.00495	0.03808	-2.6309	-2.9469	-0.39649	15	-2.4956

H.1 Results for Testing Data and New Input Dataset

The following are plots of the output of the simulation model and the neural network for the same input dataset. The first plot for each network is for the training set, while the second plot is for a test set and the corresponding network output.

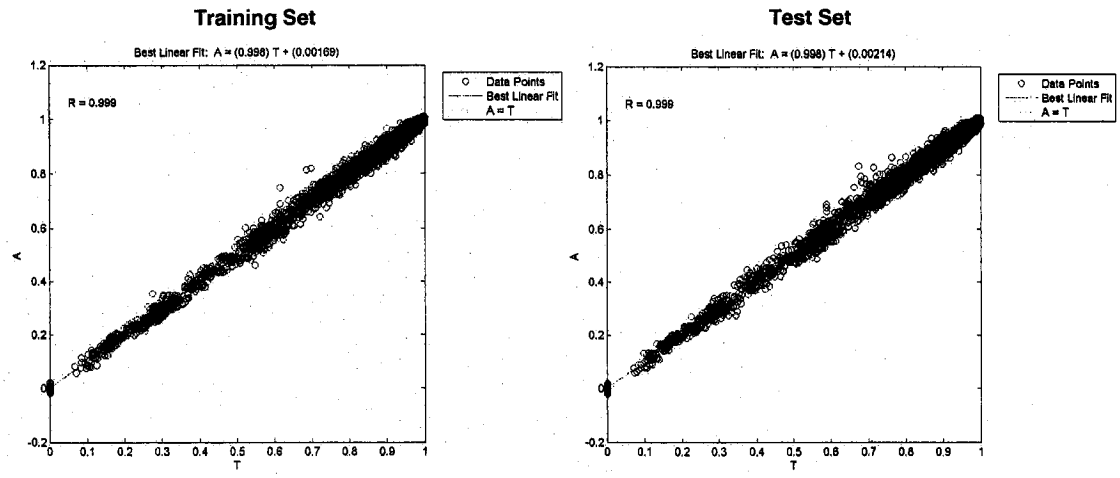

Figure H.1: Simulation Results vs. Network Results, Fill Rate, Model D

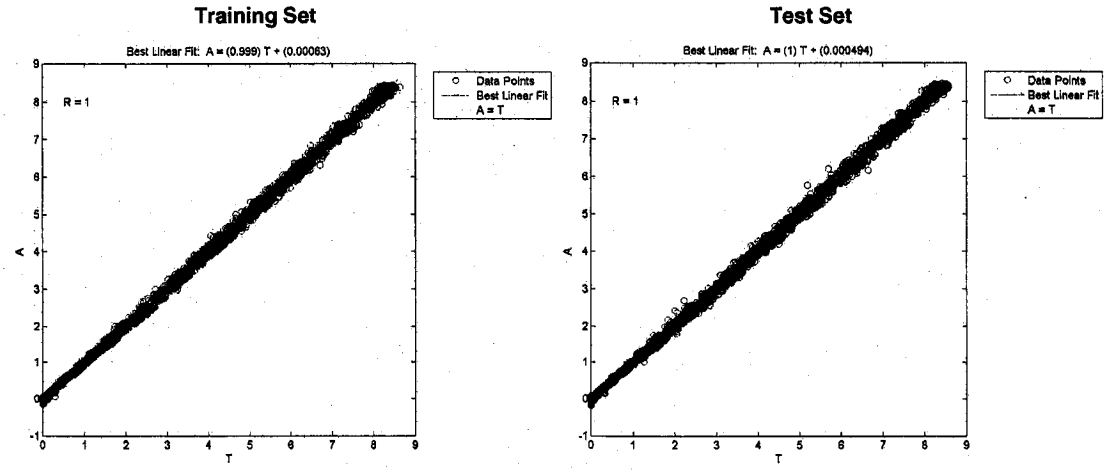

Figure H.2: Simulation Results vs. Network Results, Finished Inventory, Model D

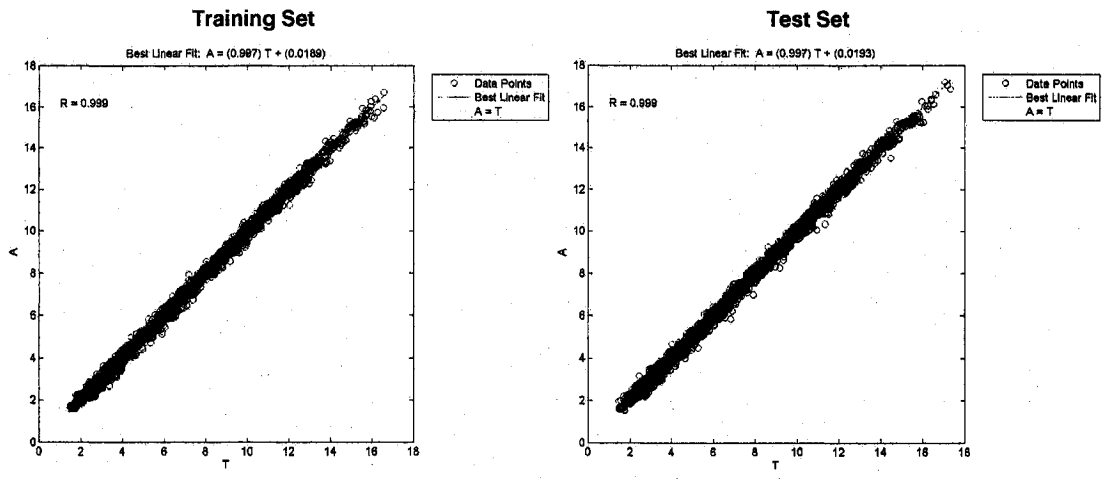

Figure H.3: Simulation Results vs. Network Results, WIP Product 2, Model D

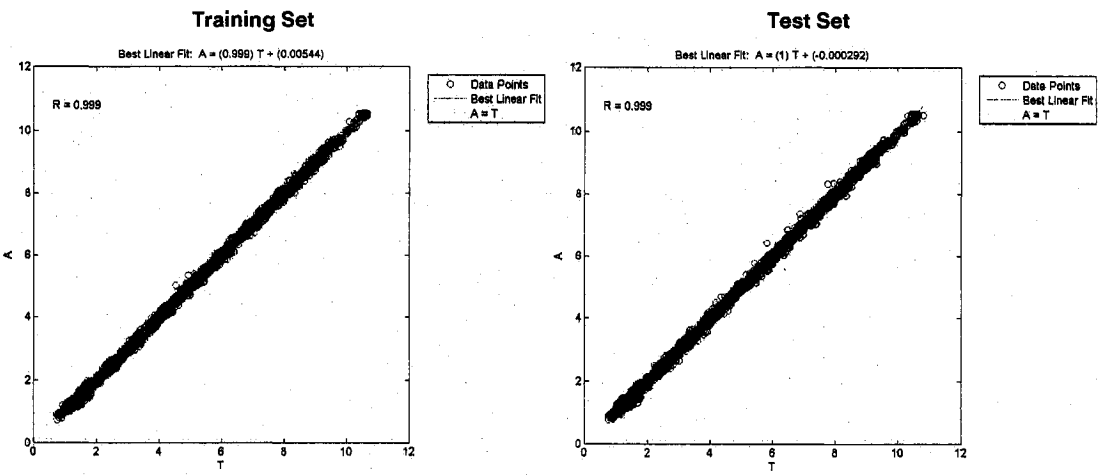

Figure H.4: Simulation Results vs. Network Results, WIP Product 3, Model D