2009

# PRODUCTION SEQUENCING AND STABILITY ANALYSIS OF A JUST-IN-TIME SYSTEM WITH SEQUENCE DEPENDENT SETUPS

John Thomas Henninger
*University of Kentucky*, tom.h@uky.edu

ABSTRACT OF DISSERTATION

John Thomas Henninger

The Graduate School

University of Kentucky

2009

# PRODUCTION SEQUENCING AND STABILITY ANALYSIS OF A JUST-IN-TIME SYSTEM WITH SEQUENCE DEPENDENT SETUPS

---

ABSTRACT OF DISSERTATION

---

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By
John Thomas Henninger

Lexington, Kentucky

Co-Directors: Dr. Kozo Saito, Professor of Mechanical Engineering
and Dr. Larry Holloway, Professor of Electrical Engineering

Lexington, Kentucky

2009

ABSTRACT OF DISSERTATION

# PRODUCTION SEQUENCING AND STABILITY ANALYSIS OF A JUST-IN-TIME SYSTEM WITH SEQUENCE DEPENDENT SETUPS

Just-In-Time (JIT) production systems is a popular area for researchers but real-world issues such as sequence dependent setups are often overlooked. This research investigates an approach for determining stability and an approach for mixed product sequencing in production systems with sequence dependent setups and buffer thresholds which signal replenishment of a given buffer. Production systems in this research operate under JIT pull production principles by producing only when demand exists and idle when no demand exists.

In the first approach, an iterative method is presented to determine stability for a multi-product production system that operates with replenishment signals and may have sequence dependent setups. In this method, a network of nodes representing machine states and arcs representing the buffer inventory levels is used to find a stable trajectory for the production system via an iterative procedure. The method determines suitable buffer levels for the production system that ensure that a trajectory originating from any point within a buffer region will always map to a point contained on another buffer region for all future mappings.

This iterative method for determining the stability of a production system was implemented using an algorithm to calculate the buffer inventory regions for all arcs in a given arc-node network. The algorithm showed favorable results for two and three product systems in which sequence dependent setups may exist.

In the second approach, a product sequencing algorithm determines a product sequence for a production system based on system parameters – setup times, buffer levels, usage rates, production rates, etc. The algorithm selects a product by evaluating the goodness of each product that has reached the replenishment threshold at the current time. The algorithm also incorporates a lookahead function that calculates the goodness for some time interval into the future. The lookahead function considers all branches of the tree of potential sequences to prevent the sequence from travelling down a dead-end branch in which the system will be unable to avoid a depleted buffer. The sequencing algorithm allows the user to weight the five terms of the goodness equations (current and lookahead) to control the behavior of the sequence.

KEYWORDS: Mixed Model Sequencing, Just-In-Time Manufacturing, Stability Control, Switched Arrival Systems, Sequence Dependent Setups

John T. Henninger
Student's Signature

Date

PRODUCTION SEQUENCING AND STABILITY ANALYSIS OF A JUST-IN-TIME SYSTEM WITH SEQUENCE DEPENDENT SETUPS

By

John Thomas Henninger

Dr. Kozo Saito
Co-Director of Dissertation

Dr. Larry Holloway
Co-Director of Dissertation

Dr. James McDonough
Director of Graduate Studies

RULES FOR THE USE OF DISSERTATIONS

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the dissertation in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this dissertation for use by its patrons is expected to secure the signature of each user.

| Name | Date |
| --- | --- |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

DISSERTATION

John Thomas Henninger

The Graduate School

University of Kentucky

2009

# PRODUCTION SEQUENCING AND STABILITY ANALYSIS OF A JUST-IN-TIME SYSTEM WITH SEQUENCE DEPENDENT SETUPS

---

DISSERTATION

---

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By
John Thomas Henninger

Lexington, Kentucky

Co-Directors: Dr. Kozo Saito, Professor of Mechanical Engineering
and Dr. Larry Holloway, Professor of Electrical Engineering

Lexington, Kentucky

2009

## Acknowledgments

This dissertation would not have been completed without the guidance, weekly meetings, and encouragement from my Committee Co-Chair Dr. Larry Holloway, who even when his schedule was full, still made it a priority to meet with his studentss. I also must thank Dr. Kozo Saito for his encouragement and patience during this long process, as well as his willingness to provide his time to be my Committee Co-Chair. I would also like to thank the remaining members of my Committee, Dr. Ibrahim Jawahir and Dr. Keith Rouch, as well as the outside examiner Dr. Alberto Corso, for their time and assistance to complete this dissertation.

## Table of Contents

## List of Tables

## List of Figures

# 1 Introduction

The purpose of this research is to investigate and propose a production system stability algorithm and a product sequencing algorithm to assist manufacturers with unavoidable sequence dependent setups. The algorithms are intended to be used to determine an effective and stable product sequence that is responsive and effective at meeting customer demand in production systems with significant and sequence dependent changeover costs. The algorithms are based on Just-In-Time (JIT) (note that lean manufacturing, Toyota Production System, and JIT will be used interchangeably in this dissertation) principles of production, meaning that production is triggered based on customer demand, not forecasted, predicted, or scheduled demand. Production is signaled to replenish the product buffer when a certain buffer threshold is reached, which pulls products through the manufacturing system as they are consumed by the customer. Conventional production systems push products through the production system into finished goods inventory based on forecasted demand and are often unresponsive to changes in customer demand.

This research originated with a plant visit to a liquid solvents packaging facility which had a high variety of products and variable customer demand. Packaging bulk liquids into smaller containers can create sequencing issues due to the chemical properties of each product. Dockx et al. [1] examined the issues involved with scheduling production in the chemical industry due to the "inevitable chaos, caused by a multitude of possible conflicting choices, and by the fuzziness and uncertainty of the parameters involved." The most common issue with bulk liquid packaging is dealing with sequence dependent changeover or setup costs between products when some chemicals are incompatible and require special changeover procedures. Although this research began with the packaging industry, the sequencing issues are not unique to this industry and are applicable to any manufacturer with a variety of products with variable demand and unique changeover costs between products.

The current state of the packaging industry varies from company to company but a common theme throughout the industry and other manufacturers is the need to reduce costs, improve efficiency, and increase market share. The implementation of lean manufacturing can help a company to meet these objectives.

## *1.1 Problem and Solution*

The first problem addressed in this research examines whether or not a stable product sequence exists for a given production system. This problem can be solved analytically or with an algorithm that will examine a JIT production system and determine if a path through the systems exists in which production can meet demand for all future time and the buffers remain positive. The analytical results are a significant contribution to this field of research because if a solution is found for the system, it is guaranteed to be stable for all future cycles through the system. The stability algorithm is a starting point for implementation of the iterative method and is a foundation upon which to build future research. The current stability algorithm requires significant computational time for production systems with more than a few products which led to addressing the second problem. The stability analysis method is intended to be used as an off-line tool to determine if a stable product sequence trajectory exists for the production system.

The second problem addressed in this dissertation focuses on the need to determine a feasible product sequence for a JIT production system with sequence dependent setups. This research proposes a straightforward heuristic sequencing algorithm that is based upon JIT principles with lookahead as a solution for determining a product sequence when setups are present. The inclusion of lookahead time into the sequencing algorithm helps to ensure a feasible solution by considering future buffer conditions to avoid dead-end paths in which a product buffer crashes. The proposed sequencing algorithm is a contribution to the field of research in that the sequencing algorithm provides an effective method for determining a feasible sequence with sequence dependent setup with JIT principles. The algorithm is intended to be practical enough to be implemented on the manufacturing floor but it is also intended to be a building block to be used by other researchers to continue the advancement of product sequencing research. The sequencing algorithm, when implemented in a production setting, would be used as an on-line sequencing tool that has an information feedback loop from the production system and finished goods inventory buffers.

## *1.2 TPS and JIT Principles*

The Toyota Production System (TPS) and JIT are often considered to be the same concept by researchers as well as practitioners, but this is not true because TPS is more than just JIT. TPS is often described as a house that is composed of tools which are combined together create a philosophical approach to manage the entire manufacturing system – people, processes, facilities, and materials. JIT is considered to be one of the columns used support the roof of the house,

which represents continuous improvement (Kaizen), see Figure 1.1. The second column is the concept of stopping to correct a problem at the source (Jidoka) and not pushing the problem any further through the system. The house is built upon a foundation of standardization, performance measures, and mutual respect. Note that without all components working together, the house will not stand.

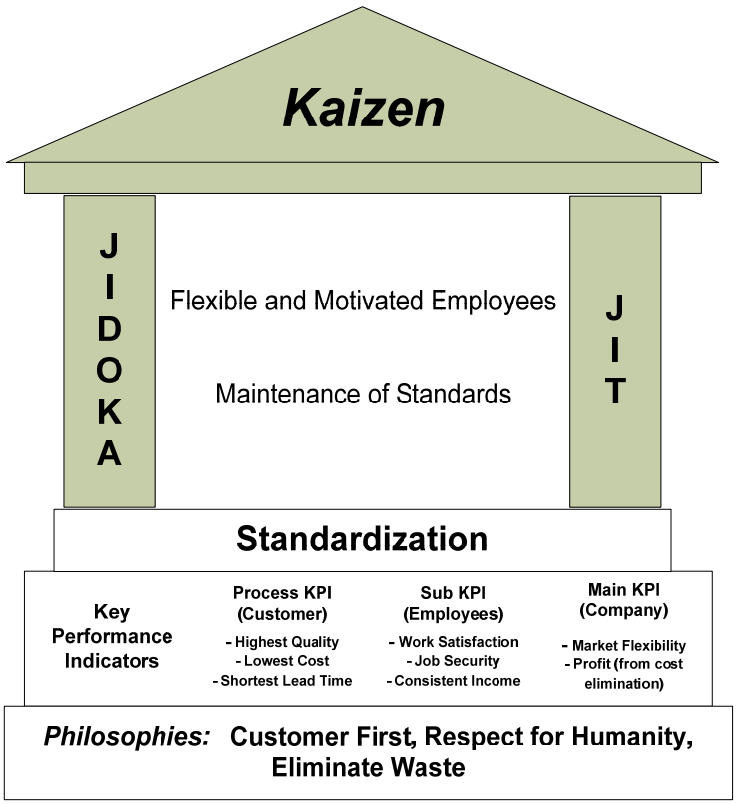


Figure 1.1: Toyota Production System House

Chemical packaging companies rely on some form of production scheduling to determine what, when, and how much to package and the scheduling can often be responsible for increased costs due to excessive levels of inventory and the inability to meet changing customer demand. Some authors emphasize the importance of flexibility and low costs to be able to enter into today's rapidly fragmenting niche markets where customers' desires are always changing [2]. Currently most companies with a high variety products use an MRP system to schedule production. This leads to the problem of very high safety stocks created by the inaccuracies within MRP systems (see MRP discussion in Appendix III). This research incorporates JIT principles that will provide a useable method to sequence products to be packaged and balance the tradeoff between a smooth mixture of products and lost time and capacity from changeovers between products.

The packaging company visited at the beginning of this research suffered from excessive inventory levels, backordered jobs, processing large batches, and very slow changeovers between products. Decreasing changeover and setup times can be accomplished by using dedicated piping, quick release fasteners, and other commonly used lean changeover methods. If quick changeovers are accomplished, the scheduling problem still remains a roadblock to the lean transition. In the pre-lean state, approximately $3.5 million of products are in inventory and the inventory typically stays in the warehouse for up to six months prior to be purchased by a customer. By contrast, a JIT system will only have enough inventory on hand to cover demand until the inventory is replenished. For example, if a raw material takes seven days to reach the facility from the supplier and three days to package, then ten days of inventory plus some amount of safety stock would be on-hand in a lean system. Current push scheduling systems do not have the intelligence to manage production scheduling and finished goods inventory (FGI) at the low levels present in a JIT system.

This research incorporates pull-production principles by triggering production only when a given product drops below a user defined buffer threshold. Production is based on the buffer level, which provides an information feedback loop to the algorithm and it pulls products through the system based on customer consumption. This is the same lean principle that is used in a signal kanban production system or a pattern production system. A signal kanban system operates with a signal placed at a given buffer level and the product enters the replenishment queue when the signal is reached. A pattern production system operates with the buffer threshold set to the maximum buffer level for all products and after refilling a product, the product immediately reenters the production queue which creates the production pattern. A signal kanban may experience idle production but a pattern production system will not experience idle time.

### *1.3 Common Liquid Packaging Issues*

The process of packaging large quantities of a liquid into smaller containers or repacking from a larger container into multiple smaller containers has certain unavoidable issues that are present regardless of what type of liquid is being packaged. Often these issues will make the transition from the typical batch and queue push production system to a JIT production system very difficult. This section will highlight these issues but a more in depth discussion can be found in Appendix I*V*.

A changeover between materials in a packing facility can be a very laborious task and if not done correctly can be very costly in the contamination of products, loss of material, employees sustaining injuries, or polluting the surrounding environment. The actual cleanout process can differ between packaging facilities but when viewed in the broader sense of the entire production system it is equivalent to a changeover in other manufacturing systems. The changeover costs may vary based on the sequence of products and the cost might be significant if the cleanout between materials is slow. The cleanout process will vary depending upon what chemical was packaged, some require a water cleanout and others require air or nitrogen, while other chemicals must be cleaned out using solvents and then water or air to remove the solvents.

Another issue of changeover between chemicals is the sequence in which some chemicals must be packaged. The sequencing of chemicals is important because some chemicals will absorb the odor from the previous product even after the cleanout process. Sequencing of chemicals is also important when a high purity level is desired because some chemical will not show contamination from the previous chemical even when the contamination is present. The necessity to sequence materials in a packaging system adds another layer of complexity to the production sequencing algorithm but this information can be captured in the changeover costs. An infinite setup cost will not allow two incompatible products to follow one another to avoid these issues.

### 1.3.1 Definition of Algorithm

The term algorithm refers to any well-defined computational procedure that takes some input set of values and produces some output set of values [3]. The term algorithm in this work refers to a mathematical model that consists of equations and constraints that are used to characterize a production system. Note that the stability algorithm could be applied to any system that can be represented with a network, not just production systems. Algorithms can be classified into two groups – deterministic and stochastic.

A stochastic algorithm is an algorithm that contains variability in the mathematical model or solution method and therefore a set of initial conditions may not always yield the same solution. A genetic algorithm is an example of a stochastic algorithm in which the solution begins with a random population and which are then evaluated based upon “fitness.” The fittest individuals are combined to create the next generation and this is repeated as the algorithm evolves toward the best solution. Due to the randomly generated initial population, a genetic algorithm with a given

set of initial conditions may not return the same final results when with repeated runs of the algorithm.

A deterministic algorithm is an algorithm in which the mathematical model contains no variability and will always yield the same results for a given set of initial conditions and the mathematical model consistently predicts the state of the system. Deterministic algorithms are often too complex to be solved for a production system when the variability of the system and conflicting choices are modeled. A deterministic production system is one in which all states are known and a given set of initial conditions will always yield the same final results. Note that this research assumes that the given production system being considered does not contain variability and is therefore a deterministic system that can be solved with a deterministic algorithm.

A heuristic approach is a method in which the solution, if found, will be a feasible solution but it is not guaranteed to be the optimum solution for the system. A heuristic is often used to find an acceptable solution to a stochastic system in which all future states are not known due to unpredictable factors (machine breakdowns, variability in customer demand, etc.) or for a deterministic system for which finding an optimal solution is to complex. A heuristic algorithm will typically include the ability to learn from past experiences and apply the artificial intelligence to future decisions. The solution proposed by Dockx et al. [1] for the chemical industry was a heuristic algorithm that incorporated a stochastic search, deductive reasoning, and artificial intelligence to replicate the human scheduler that has intuition from years of experience and has learned from past mistakes.

This research considers a deterministic algorithm that employs a heuristic method for sequencing mixed products for a given production system. The sequencing algorithm uses a weighted goodness calculation with or without lookahead to determine a feasible production sequence. The stability algorithm uses an iterative method to yield analytical results for a given production system. If a solution is found by the stability algorithm, the solution will guarantee a stable production trajectory for the production system.

### 1.3.2 Definition of Stability

The issue of buffer stability is an important problem that must be addressed in manufacturing systems. A buffer can be a very effective means to aid the production system by smoothing fluctuations caused by variability which can come from processes, breakdowns, workers,

transportation, etc. The proper sizing of a buffer is a key component of TPS, in that the properly sized buffer will motivate workers, reduce inventory costs, and prevent nervousness of the system. Consider a workstation that has a very large buffer (two shifts worth of production) upstream and downstream of the station. The worker must work non-stop for two entire shifts before depleting the upstream buffer or filling the downstream buffer. Over time the worker may become demoralized and unmotivated and eventually the quality and speed of work will decrease. A large buffer can also mask system issues such as breakdowns or an improper procedure because the large buffer is never depleted to expose the issue to allow the root cause to be found and corrected. A buffer that is too small can also negatively affect a worker; consider a workstation with a buffer sized to a single product of inventory both upstream and downstream. If any disturbance occurs such as a breakdown or maintenance, the downstream buffer will be exhausted and the upstream buffer will quickly fill and block production from upstream stations. The ideal buffer level is one in which system variation does not crash the system and the worker is slightly stressed to maintain motivation, efficiency, and quality of work. When a buffer is properly sized it is a win-win relationship to manage both human and system variation according to Fujio Cho's buffer principle [4]. MRP systems typically have large buffers with no concern for the win-win relationship while JIT systems are more focused on maintaining a win-win relationship between the humans and production system.

The term stability has many different definitions depending upon the researcher, type of system, and goals of the research. This dissertation considers a production system operating over some time interval to be stable if over the time interval one of the following conditions exists:

1. A given buffer state of the production system is revisited with a finite bound between visits, meaning that the buffer state exhibits periodicity [5].
2. The buffer state of the production system remains within the bounds of $\pm\Delta$ and positive for all time [6].

Chase et al. [5] began the research on the concept of periodicity and chaos in a production system represented by a switched flow system. A graphical representation of the first condition of stability can be seen in Figure 1.2. This diagram is a simplistic three product system with the sum of the buffers equaling one, equal production rates, and usage rates sum to the production rate, which allows the system to be considered closed, and the quantity of products flowing out is equivalent to the quantity flowing in. Each buffer is characterized by $X_i$. The $X_i$ edge of the

triangle represents an empty buffer for product *i*. Each apex of the triangle represents the location where two buffers are simultaneously empty. If the trajectory reaches an apex, the system will crash and therefore each apex is a location of instability. The buffer state trajectory shown simply orbits from product to product replenishing each buffer in a periodic pattern – 1-2-1-3. Therefore the system is stable because the buffer state of the system returns to the initial starting state when the 1-2-1-3 pattern is started again (after every four cycles, regardless of the starting product). Assuming that the system is deterministic, the periodic orbit is revisited after a given finite time interval *T*, which is simply time to refill each product in the periodic pattern.

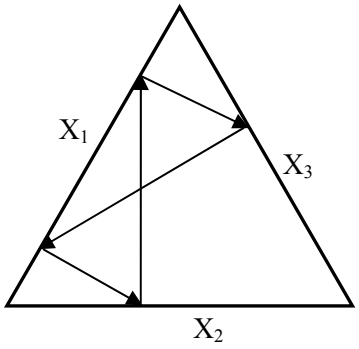


Figure 1.2: Periodic Buffer State

The concept of stability with bounded variation has been extensively researched over the years, first by Perkins and Kumar [6] and later in JIT systems by Seidman and Holloway [7-8]. An example of the second condition for stability can be seen in Figure 1.3. In this diagram the product buffer fluctuates over time but never crosses the maximum and minimum bounds represented by the dashed lines. Therefore the product is stable because it remains within the bounds over the given time interval. This stability condition requires the lower limit to be greater than zero products (an empty buffer). In some cases it may be acceptable to allow a product buffer to reach zero, perhaps just as production of the product starts, but an empty buffer is avoided in this research.

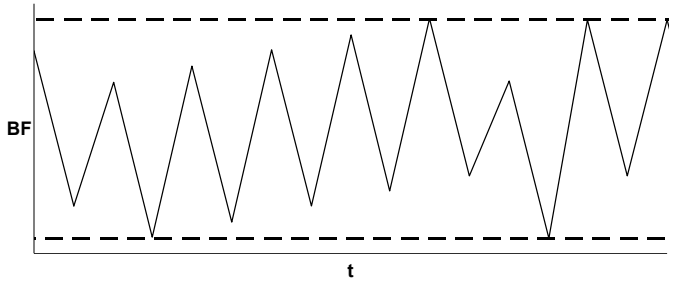


Figure 1.3: Bounded Variation

In this dissertation the upper limit of the buffer is bounded by the defined level of a full buffer for each product, therefore the upper bound is always guaranteed to exist. The lower bound must be checked to ensure that the system does not experience one or more empty buffers. An empty buffer is assumed to represent a system crash and therefore any buffer state trajectory that contains an empty buffer is considered to be unstable.

## *1.4 Overview of Dissertation*

This dissertation begins with a general introduction and overview of key issues and principles that will be addressed in subsequent chapters. Chapter 2 provides a review of previous work in the area of product sequencing for JIT systems as well as stability and chaos control in production systems. Chapter 3 discusses analytical analysis and results for stability in a given arc-node network. Chapter 4 presents an algorithm to determine the stable regions for a given arc-node network based on the analytical results. Chapter 5 provides a discussion of results from example problems using the stability algorithm. Chapter 6 discusses the development of a product sequencing algorithm with lookahead and product dependent setups. Chapter 7 provides a discussion of conclusions from the work done in this dissertation and areas of future work.

# 2 Literature Review

This review of previous work in the area is divided into two areas for clarity; production scheduling is first, followed by stability in a production system.

## *2.1 Production Sequencing*

Research in the area of production scheduling began in the 1950s with researchers such as Johnson [9], Smith [10], and Jackson [11]. Since that time production scheduling has been a popular area for researchers focused on push production, but in the last few decades Just-In-Time (JIT) production systems were introduced into this research field. In 1999 Ying and Liao [12] conducted a survey of 130 research papers that focused on production systems with setup times. Only a few of the systems considered in the survey can be classified as JIT production systems. JIT production systems have become much more popular with researchers in the past decade, but systems with setups are still seldom considered by researchers. Boysen et al. [13] conducted a survey in 2009 of over 200 research papers in the area of production sequencing of mixed-model assembly lines. In this survey, only a small fraction of the papers, around 30 of the 200+ papers considered setups and changeovers between products, which highlights that sequencing of JIT systems with setups is an area that needs further research.

Mixed-model production sequencing can be applied to a wide variety of production systems depending upon what the interests of the researcher. Research of sequence dependent setups has been applied to single machines, parallel machines, flow lines, and job shops [14]. During the process of reviewing previous work in the area of production scheduling and sequencing, it has become apparent that there are distinct divisions in the research of mixed-model sequencing for assembly lines. The first group refers to the problem as mixed-model sequencing and focuses on minimizing work overload at an assembly station, minimizing setups, and considers worker movement, station borders, cycle time, and other system parameters. The second group refers to the problem as car sequencing, which is a simplified approach to the mixed-model sequencing problem that focuses on minimizing work content overload at final assembly line stations by constraining the ratio of work intensive products [13]. The third group characterizes the problem as level scheduling or level sequencings and the researchers focus on level usage of parts or materials for the mixed-model sequence, which was originally proposed by Monden [15] as a key component of the Toyota Production System and other JIT manufacturers in the United States. The final group defines the problem using a hybrid-model sequencing system that incorporates aspects of the previous three groups, researchers such as McMullen [16] who simultaneously

considered level scheduling while minimizing the number of setups fall into this group. This dissertation is considered to be part of the work in the hybrid-model sequencing.

Mixed-model assembly line problems have the same three basic characteristics or constraints as the foundation for the mathematical model to represent the system. Consider a mixed-model assembly line with a planning horizon of $T$ time periods, where $t = 1,\ldots,T$ and the system has $N$ different models where the demand for model $i$ is $d_i$ for all $i \in N$. Let $k$ denote the stage of the sequence, for example consider a given sequence A-B-C, $k$ is stage number 1, 2, or 3. Note that the time $t$ is interchangeable with the stage of the sequence $k$ if only one model is produced during each time period.

- The first characteristic is a binary variable to record what model $i$ is produced during a given cycle $t$, which is represented by $x_{i,t}$, which is either 0 (not produced) or 1 (model produced) for all models $i \in N$ at each time period $t$.
- The second constraint of the mathematic model is to ensure that only one model $i$ is produced during a given time $t$ (or stage $k$, where $k = t$), which is accomplished with $\sum_{i=1}^{N} x_{i,t} = 1$ for all time $t = 1,\ldots,T$.
- The third constraint is that demand $d_i$ for a model $i$ is produced for all models $N$ during the planning horizon $T$, where $\sum_{t=1}^{T} x_{i,t} = d_i$ and $\sum_{i=1}^{N} d_i = D_T$.

### 2.1.1 Mixed-Model Assembly Line

Mixed-model (MM) sequencing problems are related to the car sequencing (CS) problems but are typically more complicated than the CS problems because more parameters are considered in the MM problems. The MM problem considers work content time or space required to complete the assembly function at a given assembly line station and the station has a defined length. An assembly station is sized based on the average capacity required for all models to be assembled which means that some models will create a work overload at a given station while other models will be completed early at a given station.

The assembly line may consist of closed, open, or a hybrid mix of boundaries for the stations. A closed boundary means that a worker can start working on the product only when it crosses into the station and must stop working on the current model if it reaches the end of the station. An unfinished product is considered work overload and can be completed by utility workers in a

closed station model or the line must be stopped. An open station model allows the worker to start working on the next product after finishing the current product (no idle time). The open station worker stops working when the product is completed or when the next product enters the assembly station depending on the definition of the system. The worker may travel beyond the borders of an open station. The MM sequencing problem characterizes the assembly line by considering the speed of the conveyor, the processing time of each model, the speed that the worker returns to the beginning of the station, cycle (takt) time of the assembly line, station length, number of stations, and model demand as parameters to accurately model the system.

The MM researchers typically develop a model and method to optimize some objective or combination of objectives. Minimizing work overload at all stations and over all production cycles is the most common objective, where work overload is the amount of time or space by which the borders of a station are exceeded. Minimizing the line length or throughput time are two objectives considered by researchers but the two objectives will yield very similar results for a system in which the line does not stop and travels at a constant speed. The time to travel through the assembly line is a function of the velocity and the length of the line. Other potential objectives are to minimize the distance travelled by workers, total idle time, duration of line stoppages, and setup costs or some combination of multiple objectives.

#### 2.1.1.1 MM Assembly Line Problems with Setups

Most authors assume that the assembly tasks or the mix of models require insignificant setups between models and are therefore ignored, but there are a few researchers who complicate the MM problem further by considering setups between products. Burns and Daganzo [17] evaluated the trade-offs between setups between jobs and production capacity utilization in determining the assembly line sequence for mixed-models. Setups were grouped into three potential areas, with the first consisting of wasted materials such as paint that is purged to changeover to a new color. The second area consists of lost labor or machine capacity due to changeovers, such as during the stoppage for a changeover of a stamping machine. The final area of setup cost is subpar product quality that occurs when changing to a new product, such as the first few injection molded parts being unacceptable after a mold change due to the mold not being at the correct temperature for the first few cycles. Other authors, such as Bolat et al. [18] considered sequence dependent setups for the MM sequencing problem and developed a heuristic procedure to concurrently minimize utility work and setup costs, but it should be noted that the setups are considered independent of the time required to complete the model. Kim et al. [19] considered the MM

problem with a hybrid mix of stations and sequence dependent setups. A genetic algorithm was developed to find a near optimum sequence with the objective to minimize the length of the assembly line. Rahimi-Vahed and Mirzaei [20] proposed a shuffled frog-leaping algorithm to simultaneously optimize three objectives of total utility work, production rate variation, and setup costs to sequence the MM system with setups. Rahimi-Vahed et al. [21] again consider the same three objectives for a MM system but propose a multi-objective scatter search algorithm to optimize the product sequence. Tavakkoli-Moghaddam and Rahimi-Vahed [22] studied the MM assembly line problem in a JIT environment using a memetic algorithm (a hybrid genetic algorithm that includes local refinement and evolution) to simultaneously optimize the sequence based on minimizing utility work, production rate variance, and setup costs. Rahimi-Vahed et al. [23] included real options of the cars with associated values to maximize the value of the product mix for a MM system. After developing the "best" product mix, a genetic algorithm and a memetic algorithm was then used to determine the optimized sequence based on minimizing the utility work, sequence dependent setups, and production rate variation. Rabbani and Rahimi-Vahed [24] consider MM problem with setups and real-world variability in demand and compare the algorithm against the goal chasing method as well as the method proposed by Miltenburg and Sinnamon [25]. Kim and Jeong [26] proposed a branch and bound method to determine the optimum sequence for MM systems that have sequence dependent setups while minimizing the total unfinished work for the sequence.

### 2.1.2 Car Sequencing Problem

The car sequencing problem was originally formulated by Parrello et al. [27] in 1986 to solve a problem for General Motors Research Laboratories. Instead of sequencing models based on detailed work content calculations as in the MM procedure, the CS problem attempts to minimize work overload by constraining the number of successive high work content models and yet still meet the required demand for each model. The capacity constraint (often referred to as a hard constraint) is a ratio such as $H_o : N_o$, which means that only $H_o$ models can contain option $o$ out of $N_o$ successive models to avoid work overload [13]. An example of a high work content model is a car that requires a sunroof or air conditioning. If the sequence of cars cannot satisfy the capacity constraint, a penalty constraint (often referred to as a soft constraint) will penalize each step of the sequence that violates the constraint. The penalty may be constant or may vary based on the number of successive violations. For example, the first violation will incur a penalty of one but the second violation is penalized a value of five. Parrello et al. optimized the sequence by minimizing the penalty cost for the sequence. Other authors such as Fliedner and Boysen [28]

optimized the sequence by minimizing number of constraint violations, while other authors [29-30] have proposed more efficient algorithms.

In 2005, Renault hosted a challenge for teams of researchers to compete against one another to create the best sequencing algorithm to optimize the tradeoffs between the final assembly sequence constraints and the minimizing setups in the paint shop [31]. The resulting algorithms created small batches in the paint shop based on similar colors while minimizing the final assembly constraint violations. Other authors [32-37] have examined the same problem of simultaneous sequencing of the paint shop and final assembly and reached similar conclusion as the Renault teams.

Consider the method used to solve the CS problem, which is to sequence the products based on minimizing work overload by imposing a capacity constraint ratio to limit high work content models. What does this do to final supply lines and tier-one and tier-two suppliers? Consider a hypothetical sequencing problem that must determine the optimum sequence for two models where model A has the standard configuration of seats and model B requires power adjustable seats. The final assembly work content is equivalent for the two models, therefore the capacity constraint ratio has no effect on determining the sequence. Without considering any other factors, the sequence could be large batches alternating between manual and power seats, which would be fine for final assembly. This sequence would cause massive demand swings for the seat supplier when the supplier would have to sequentially produce dozens of the power seats, which have much higher work content as a subassembly than the manual seats. To overcome this problem, many researchers consider level sequencing which creates the sequence based on level material usage.

### 2.1.3 Level Sequencing

The level sequencing problem was first published by Monden [15], who discussed the Goal Chasing Method that was in use at Toyota to schedule the final production assembly lines. Monden defined two goals of the optimum final assembly sequence:

1. Minimize the workload variation at each stage on the assembly line.
2. Maintain a constant usage rate of every part consumed on the assembly line.

This first goal acknowledges that each workstation does not have the same cycle time on the assembly line but smoothed variation between stations will help to eliminate inefficiencies such as idle time or excessive work content for a given takt time. The second goal attempts to

maintain a smooth usage rate of materials per unit time for each model on the assembly line. To solve these problems at Toyota, the Goal Chasing I and II methods were developed [20].

Miltenburg [38] examined the problem of creating a level and balanced schedule for an assembly line with mixed-model products. Miltenburg's work is an extension of the Goal Chasing Method by Monden [15]. This paper is often cited by other researchers and is considered to be the starting point for a large portion of research dealing with level sequencing of mixed-model assembly lines. In this work the author assumed that setup times were negligible between parts, which allows the emphasis to be placed on finding an ideal sequence that provides a constant rate of usage for all parts. Keeping a constant rate of usage for every part used by the line means that if the line produced three products, A, B, and C, and if product demands are equal, an ideal constant rate of usage would be to repeat A-B-C sequence. The objective is to schedule the assembly line such that the proportion of a product produced over the total production period is close to the proportion of the demand for the given product to the total demand for all products. To accomplish this objective three algorithms and two heuristics were developed to minimize Equation 1.

$$U = \sum_{k=1}^{D_T} \sum_{i=1}^{n} \left( x_{i,k} - k d_i / D_T \right)^2 \tag{1}$$

where

$n$ = the number of products to be assembled in the line,

$D_T$ = the total number of units for all products,

$d_i$ = demand for product $i$, $i$ = 1,2,…,$n$,

$x_{i,k}$ = total number of units of product $i$ produced over stages 1 to $k$, where $k$ = 1,2,…, $D_T$

Miltenburg and Sinnamon [25] further extend the previous Miltenburg research to be applicable to multi-level JIT systems, examining issues that are present when products require differing amounts of parts and subassemblies. Algorithms and heuristics are proposed to solve the problem allowing the user to define different weights to be applied to products, sub-assemblies, components, and raw materials. This work also includes a lookahead feature that allows the algorithm to consider the current stage and the next stage as well to calculate the usage rate variation of the two stages together. The product with the lowest combined variation for the current stage $k$ and the subsequent stage $k$ + 1 is selected. Miltenburg and Sinnamon [39] extended this method to a multi-level production system that considered four levels of production

– product, subassembly, component, and raw material. The authors proposed three algorithms and three heuristics to provide the optimum sequence.

Miltenburg and Sinnamon [40] revisit their previous work and provide insight into the use of the weights in the original research and the implications on a JIT production system. The setting of the weight values allows the scheduler to vary the schedule to meet certain issues, such as if a shortage of a component occurs; the subassemblies would be scheduled as late as possible to allow time to catch up the shortage.

Ding and Cheng [41] continue the advancement of research in the area of scheduling and sequencing mixed-models on an assembly line for a JIT. The authors simplify the algorithms and heuristics of Miltenburg [38] to decrease CPU time and provide equivalent results. The proposed algorithm is a five-step process that can be computed in a single do-loop and requires significantly less computer memory. The algorithm is benchmarked against Miltenburg's work by comparing the mean squared and absolute deviation of the results over nine problems. The problem sets include 25 problems with a random production quantity between 0 and 9 per model. The results show that the deviations of the new algorithm are equivalent to the previous work of Miltenburg, but the advantage is that the computational time is significantly less. As the size of the problem sets increase the reduction in CPU time increases from a minimum value of 35% for a problem set with five products up to a 99.1% reduction for a problem set with 50 products.

This dissertation considers level scheduling as the starting point or fundamental concepts upon which to build a hybrid-model sequencing algorithm. There are many other authors that have made significant contributions in the area of level sequencing, such as Aigbedo [42-44], Kubiak [45-48], Steiner [49-50], and Sumichrast [51-53]. These authors continue the work of Monden and Miltenburg, but their contributions are considered to fall outside the scope of this dissertation.

### 2.1.4 Hybrid-Model Sequencing

This section focuses on hybrid mixed-model sequencing research. This first sub-section considers sequencing of production systems with setups or sequence dependent setups followed by a sub-section that reviews research that considers lookahead sequencing.

#### 2.1.4.1 Sequencing with Setups

McMullen [16, 54] examines the level sequencing of a JIT assembly line system with mixed-models and non-negligible setup times. Although previous discussed work [25, 38, 40-41] focused solely on the smooth material usage for a given production sequence, McMullen's work attempts to provide reasonable levels of material usage rate and the total changeover time. This research assumes that the setup times, though non-negligible, are limited to 20% of the actual time required to process the product. This research develops equation (3) as an objective function with weights for the product usage rate, equation (1), and the cumulative number of setups, equation (2), to be minimized using a Tabu Search algorithm. Miltenburg's part usage rate and Ding and Cheng's algorithms are used in this research.

The number of setups for a sequence is computed as follows:

$$S = \sum_{k=1}^{D_T} s_k \tag{2}$$

where $s_k$= 1 if the product in position $k$ is different than the product in position $k$ – 1, or 0 otherwise. The composite objective function used to consider sequence smoothness and number of changeovers is computed as follows:

$$\min: z = w_s S + w_u U \tag{3}$$

where $w_s$ is the weight applied to the number of setups and $w_u$ is the weight applied to the usage rate variation.

McMullen and Frazier [55] examine the use of simulated annealing to find a near optimum solution to the mixed-model sequencing problem. This research develops a simulated annealing algorithm and compares the results to a Tabu Search heuristic. The simulated annealing algorithm begins by generating an initial solution and the solution is evaluated using an objective function, equation (3). The next step is to generate a neighboring solution, this research uses pairwise swapping, meaning that two unique products are randomly selected and swapped. The new test solution is compared with the current solution and the better solution is kept. A calculation is performed to determine the probability of accepting an inferior solution; inferior

solutions are accepted to keep the algorithm from being trapped in a local minimum. This process is repeated for a defined number of iterations and then the temperature is lowered by the defined cooling rate and the process is repeated until the final temperature is reached. The proposed algorithm is evaluated using computer simulation and the results show that the algorithm out performs the Tabu Search heuristic.

McMullen et al. [56] further examined mixed-model job sequencing and examined the use of a genetic algorithm to find a near optimization of the system. The equations (1), (2), and (3) were again used in this research to be minimized by the genetic algorithm. The genetic algorithm functions by generating some number of initial solutions and determining the "fittest" of these solutions. A crossover is performed on the best solutions to create a new generation of solutions. The new offspring then undergo a mutation with a defined probability and the "fittest" of this generation will undergo a crossover and mutation and the process will repeat until the defined number of generations has been produced. The algorithm proposed in this research is evaluated using test cases to compare it to a Tabu Search heuristic and Simulated Annealing algorithm. The results show that the genetic algorithm and simulated annealing algorithm provide nearly equivalent results, with the only difference being an increase in CPU time for the genetic algorithm. This is caused by the genetic algorithm manipulating a large portion of the production sequence during the crossover and mutation steps. McMullen considered other optimization algorithms and methods in other works [57-59].

Mansouri [60] continue the work of McMullen utilizing a multi-objective genetic algorithm to minimize the number of setup while minimizing the production rate variation of the sequence. The author proposes that smoothing the variation of production rates can be considered a substitute for Miltenburg's method of smoothing the material usage rates. Note that both authors are assuming that all products being considered require approximately the same number and mix of parts.

Mohammadi and Ozbayarak [61] developed a method to sequence mixed-model products in a JIT environment that included setups between different models. This work built upon Miltenburg's level material usage and incorporated a setup cost function that calculated setup cost at each station of the assembly line. This work is similar to McMullen' work although the cost function (equation 4) is more complicated in that it considers sequence dependent costs instead of a generic changeover cost.

$$\min \sum_{i=1}^{N} \sum_{k=1}^{D_T} \sum_{s=1}^{S} \sum_{l=1}^{N} x_{k,i,l} C_{s,i,l} \tag{4}$$

where $C_{s,i,l}$ is the setup cost to change from model $i$ to model $l$ at station $s$ and $x_{k,i,l}$ is 1 if models $i$ and $l$ are assigned to position $k$ and $k + 1$ respectively in the sequence, otherwise $x_{k,i,l}$ is zero. This method provides optimum or near-optimum results with very little CPU time required to find the solution. The authors also compare their method to previously publish problem sets with favorable results.

Ponnambalam et al. [62] consider the multi-level (product, subassembly, component, and raw material) level rate of usage sequencing problem and method proposed by Miltenburg and Sinnamon [39]. The authors framed the problem as a MM problem instead of a level sequencing problem. The authors propose using a genetic algorithm to optimize the sequence based on the common three objectives of MM problems – minimizing utility work, minimizing variation in part usage, and minimizing setup costs. Their method was compared against Miltenburg and Sinnamon's method and proved to be an effective alternative.

Ahmadi and Matsuo [63] proposed developing families or groups of products and processing each group on a dedicated mini-line to minimize large setups for the printed circuit board industry. The method allows pull production on the lines after the groups have been established and the method also allows small setups within the group to occur. This concept could be introduced into the liquid packaging industry by the introduction of dedicated plumbing and quick cleanout packaging lines.

Doganis and Sarimveis [64] investigated the optimal scheduling for a yogurt production line using mixed integer linear programming. Yogurt production has many common issues experienced by the chemical industry due to sequence dependent setups based on fat content and flavors of the various yogurt products. As an example consider that plain or low-fat yogurt requires a very thorough cleanout prior to packaging if the previous product was high fat or flavored yogurt, but high fat or flavored yogurt following plain or low-fat products requires minimum cleanout. The yogurt industry also must consider production that matches demand due to spoilage issues if the product is not consumed in a short window of time. The proposed algorithm schedules production based on changeover cost, inventory cost, and labor cost.

Dockx et al. [1] developed an interactive scheduling program, referred to as SKYE. The program integrates deductive techniques, such as constraint propagation, with stochastic techniques, such as Simulated Annealing or Tabu Search, and temporal reasoning to schedule production for the chemical process industry. They attempt to overcome the "inevitable chaos, caused by a multitude of possible conflicting choices, and by the fuzziness and uncertainty of the parameters involved" by allowing the user to interact with the program to improve the schedule. SKYE is intended to be the scheduling module within a larger production scheduling software package that maintains other important inventory and production information.

### 2.1.5 Lookahead Scheduling

The use of lookahead for determining a production sequence or schedule has been studied by many researchers. The goal of lookahead scheduling is to forecast future conditions some time or number of products into the future and then use that information to optimize some objective function. Lookahead is also very useful to aid in the avoidance of dead-end sequences in which the sequence eventually lead to an illegal system state [65], such as empty buffers or missed demand.

The work of Miltenburg and Sinnamon [25] includes a lookahead feature that allows the algorithm to calculate the usage rate variation of the current stage and subsequent stage to choose the smoothest sequence. Miltenburg and Goldstein [66] continued the Miltenburg and Sinnamon work by developing a weighted sum objective function for multi-level smoothing (products, parts/sub-assemblies, components, materials) and workload smoothing. The problem is solved using a mixed integer program which considers a two-stage look-ahead heuristic to determine the sequence. Leu et al. [67] build upon the level sequencing method using the lookahead concept introduced by Miltenburg and Sinnamon [25] with a beam search algorithm. This work expands the solution search space to improve the probability of finding the optimum sequence. The proposed beam search method proved to be clearly superior to the goal chasing method as well as Miltenburg and Sinnamon's method.

Briant et al. [68] proposed using lookahead to aid in the batching of cars for the paint shop to solve the car sequencing problem. The authors' approach was to initially lookahead in the sequence of up to six cars to batch common colored cars to decrease setups in the paint shop. As the sequence progressed, the lookahead distance decreased or was no longer used to encourage

the batching early in the sequence. The authors' results were shown to be comparable to other researchers.

Gupta and Sivakumar [69] examine lookahead batching for delivery of products to a semiconductor production system that operates under JIT principles. The authors developed a method to control delivery performance with the objective of minimizing earliness and tardiness measures. The approach is a combination of scheduling and discrete event simulation in which decisions are made based on current information and simulated future conditions. The authors report favorable results using the method for given test cases.

#### 2.1.5.1 Discrete Event Systems

Discrete event systems (DES) are dynamic systems that evolve due to random occurrences of discrete qualitative events; manufacturing systems are often framed as a DES. Supervisory control theory developed by Ramadge and Wonham [70] is often used to solve DES problems. The approach for supervisory control is to synthesize the supervisor offline of complete models of the system behavior, but it can be difficult to construct complete models when behaviors are complex or vary over time. The limited lookahead policy (LLP) was proposed by Chung et al. [71] as less computationally intense method to overcome the difficulties of the offline approach of calculating all possibilities. The authors' LLP approach determines the next control action by projecting the system behavior into the future N-steps, which is the lookahead window. This is repeated after the execution of each event. The LLP control action is calculated with either a conservative attitude or an optimistic attitude. The conservative attitude tends to result in a restrictive control policy while the optimistic attitude may result in violation of constraints of the system. Chung et al. [72] continued the previous work using the LLP, but developed a recursive procedure that makes use of previous calculations of trees for the next tree pair to greatly reduce computational time.

Kumar et al. [73] extended the LLP work of Chung et al. by developing an extension based limited lookahead (ELL) supervisor. The ELL determines the next control action by estimating the future behavior of the system as well as using knowledge of the system, such as an upper bound on the tree length of uncontrollable events. The ELL is more permissive than the LLP proposed by Chung et al. and the ELL also has more relaxed assumptions.

Many other authors have continued examining lookahead methods to control DES. Takai [74] examined the Kumar et al. ELL method more in depth and determined that in some conditions it is equivalent to the conservative attitude LLP method. Cho and Lim [65] were concerned with optimal behavior of the system and proposed the online tracing supervisory control method. This method showed favorable results when compared to the offline method.

## *2.2 Stability of Production Systems*

Production sequencing is a key factor for the production system to meet demand, but an equally important aspect that must be considered is the long term stability of the production system to ensure that the system can meet demand. Stability in this dissertation is considered as bounded inventory with no backlog in which the buffer inventory levels remain positive over all time.

Seidman and Holloway [7] examined stability in a pull production system that operates using signal kanbans, which are equivalent to a low inventory replenishment threshold. The researchers developed a method to set the reorder point such that there is no backorder queue and the buffers remain positive. The method is further extended to pattern production to determine the minimum buffer level to ensure no backorders occur.

Seidman and Holloway [8] further examine stability in pull production systems by considering control methods when significant setups are present. Signal kanbans with either fixed-fill levels or fixed-batch size, as well as pattern production were the two methods examined. The fixed-fill signal kanban method performs better than the fixed batch signal kanban. The authors note that the fixed-fill variant is equivalent to the switched arrival system which was shown by Chase et al. [5] to be chaotic in nature. The authors conclude that the fixed-fill variant performs better (lower average inventory) than the fixed-batch signal kanban due to long-term cyclic behavior than can occur in the fixed-batch policy. Also the pattern production method performs better than fixed-fill signal kanban by requiring less inventory for the same level of service.

### 2.2.1 Switched Arrival Systems

Switched arrival systems (SAS) have been used to model a wide variety of systems since the late 1980s, including manufacturing, data networks, and fluid flow. The SAS is composed of a server (such a manufacturing station) that is responsible for replenishing multiple buffers (inventory of products). The server replenishes a buffer until a threshold is reached at one of the other buffers (threshold of zero is an empty buffer) at which point the server switches instantaneously to the

new buffer. This switching of the server occurs each time a threshold is reached at another buffer. The system evolves over time to create a state trajectory through the system buffers. A pull production system in which a server replenishes buffers is considered to be a SAS.

Chase et al. [5] examine periodicity and chaos in both SASs and switched server systems for a case of three products. Note that the authors employed several assumptions to aid in the analysis of the systems: fixed fill rate of one and cumulative consumption rate of one to create a closed loop system. The authors note that the SAS has sensitive dependence on initial conditions and is chaotic in nature and it is highly unlikely to settle into a periodic orbit.

Horn and Ramadge [75] introduced the concept of thresholds for a SAS and studied their affect on the dynamic behavior of a three product system. The threshold functions as a bound on the buffer, whether an upper limit or lower limit. The lower limit threshold does not change the dynamic behavior of the system from the original system because a lower limit is equivalent to rescaling the system. Upper limits were studied (with lower limits set to zero) and the server prematurely stops filling the buffer when the upper limit is reached. The server then switches to the buffer with the least amount of work. The authors determined ranges of values in which the upper limits will result in periodic stable behavior of the SAS, when using one, two or three limits. Three limits provide the best results by preventing the state trajectory from reaching the unstable location where two products are simultaneously empty.

Ushio et al. [76] re-examine the three product SAS in an attempt to control the chaotic behavior, while applying the same assumptions that were used by Chase et al. The authors built upon the work of Ueda et al. [77] who showed numerically that chaos and periodic orbits can occur when the processing time is limited to a prescribed value. The authors propose a control method to stabilize the unstable periodic orbits by limiting the continuous processing time in the SAS. The authors show that their method produces stable periodic orbits. Li and Ushio [78] examined controlling chaos in a three buffer SAS by implementing flow connections between buffers exist. This method was shown to be effective at producing periodic orbits. Tian [79] proposed using a time-delayed impulsive feedback method for detection of unstable periodic orbits embedded in a chaotic system.

Ushio et al. [80] continue the previous chaos control work by Ushio et al. [76] and extend the method to a SAS with $N$ buffers. The method is the same as the previous work in which chaos is

controlled by limiting the processing time for a given buffer, but the system is generalize to consist of *N* buffers. The authors show that this method again performs well for the larger SAS.

# 3 Node Network Analysis

## *3.1 Introduction*

In the most general sense the node-arc network used in this research simply consists of information (a region) stored on each arc that is transformed by a function within the node that may or may not be unique at each node. The newly transformed information is intersected with the information stored on another arc attached to the node. The information may be transformed forward by the function of the node or may be pseudo-inversely transformed backward through the node.

The node-arc network is used extensively in this research to evaluate the stability of a production system. The system that is being evaluated by the stability algorithm is a network of nodes that are connected together by arcs. Each arc contains one or more sets of data (the rectangles in the diagram below), currently in the form of a matrix, that can be operated upon by the upstream or downstream node. Each node is host to a transformation function which may differ from node to node depending upon which type of node is defined for a given node. A node represents a processing state of the system and the transformation function defines the method in which the inventory changes while the system is in the given processing state. For example a node may indicate production of one product, or setup for production of that product, or only consumption of all products.

The intent of the method is to cycle through the network updating the upstream or downstream regions adjacent to each node based on the downstream or upstream regions until all regions no longer change when updated. A solution for the system is found when the updated region is equivalent to region prior to being updated on all arcs.

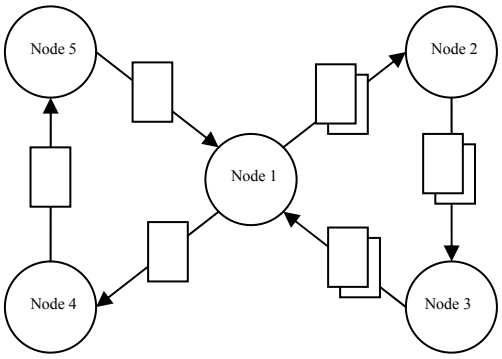

Figure 3.1: Arc-Node Network Model

### *3.2 Model Description*

Let $k$ be the number of products in the production system. The inventory of all products at the given time is represented by a vector of dimension $k$, where the $i^{th}$ row of the vector is the inventory of the $i^{th}$ product. The following notation is used for this dissertation:

- $s, q, x, y, z \in \mathbf{R}^k$ represent vectors indicating inventory of the products
- $S, Q, X, Y, Z \subseteq \mathbf{R}^k$ represent regions of inventory vectors

Thus for example consider some $X \subseteq \mathbf{R}^k$, which the elements of $\boldsymbol{X}$ are inventory vectors $x$ where $x \in X$.

The production system model is described as:

- a set $N$ of nodes
- a set of directed arcs $A \subseteq N \times N$ connecting the nodes, where every node has at least one incoming arc and at least one outgoing arc.
- a transformation function $f_n(\bullet)$ for each node $n \in N$, where for some vector $a \in \mathbf{R}^k$, the function $f_n(a)$ yields a set (possibly empty) of vectors in $\boldsymbol{R^k}$.

Consider an arc $a = (n_1, n_2)$, between two nodes $n_1$ and $n_2$, which has a set of points $Z(a) \subseteq \mathbf{R}^k$ associated with it. $Z(a)$ may be empty or may represent one or more disconnected regions. Thus any given network can be described by

$$\eta = (N, A, f(\bullet), Z(\bullet))$$

where $N$ is the set of nodes, $A \subseteq N \times N$ is the set of arcs, $f(\bullet)$ represents the transformation functions at each node, and $Z(a)$ represents the region mapped to each arc $a \in A$.

For clarity, the remainder of this chapter will use a localized notation, where a set of inventory vectors entering a node $n$ is $X_n$, and a set of inventory vectors leaving node $n$ is $Y_n$. Given a node $n$, the set of directed arcs leading to node $n$ are denoted as $Arc_{in}(n)$ and the set of directed arcs originating from $n$ are denoted as $Arc_{out}(n)$ (i.e. $Arc_{in}(n) = \{a(n',n) \in A \text{ for some } n' \in N\}$ and $Arc_{out}(n) = \{a(n,n') \in A \text{ for some } n' \in N\}$). The localized $X_n$ represents the set of all regions associated with all arcs contained in $a_{in}(n)$, such that

$$X_n = \bigcup_{a \in Arc_{in}(n)} \left(Z(a)\right).$$

The localized $Y_n$ similarly represents the set of all regions associated with all arcs contained within $a_{out}(n)$, such that

$$Y_n = \bigcup_{a \in Arc_{out}(n)} \left(Z(a)\right).$$

### 3.2.1 Basic Arc Type

An arc is simply a means of conveying and directing information between the nodes in the network. The information stored in any arc in the system is one or more regions. The regions are unchanged by the arc and are only transformed by a node. An arc also allows the direction of flow to be defined between each node because the arcs used for this node network are unidirectional. There is always one or more arc(s) into a given node and one or more arcs out of a given node.

### 3.2.2 Basic Node Type

The general node consists of a set of one or more functions that transform an incoming region to an outgoing region and inversely transform an outgoing region to an incoming region.

## *3.3 Node Transformation Functions*

At each node a function $f(x)$ or pseudo-inverse function $f^{-1}(x)$ is applied to transform a region(s) on one side of the node to create one or more regions on the other side of the node, see diagram below. The inverse function is referred to as a pseudo-inverse function because the inverse functions used in this research may or may not be one-to-one functions.


Figure 3.2: Transformation of Regions

Given a node such as in the diagram above, the updated region, $Y'$, is found by transforming the $X$ region and intersecting the transformed region with the $Y$ region, such that

$$Y' = f(X) \bigcap Y .$$

The updated region, $X'$, is found by inversely transforming the $Y$ region and intersecting it with the $X$ region, such that

$$X' = f^{-1}(Y) \bigcap X .$$

### 3.3.1 Definition of Transformation Function and Inverse Function

Given a transformation function *f(•)*, which operates over vectors in $\mathbf{R}^k$ , the function is generalized over a set $S \subseteq \mathbf{R}^k$ , which is composed of elements $s \in \mathbf{R}^k$ , as follows:

$$f(S) \coloneqq \bigcup_{s \in S} f(s) .$$

Note that $f(\varnothing) \coloneqq \varnothing$ . Let the pseudo-inverse function for some $q$, which is an element of $f(s)$, be defined such that

$$f^{-1}(q) \coloneqq \{s \mid q \in f(s)\} .$$

The pseudo-inverse transformation function generalized over set $B \subseteq \mathbf{R}^k$ , which is composed of elements $b \in \mathbf{R}^k$ , as follows:

$$f^{-1}(B) \coloneqq \bigcup_{b \in B} f^{-1}(b) .$$

Using this generalization it can also be shown that given sets $S \subseteq \mathbf{R}^k$ and $B \subseteq \mathbf{R}^k$,

$$f^{-1}(S \bigcap B) = f^{-1}(S) \bigcap f^{-1}(B).$$

Note that $f^{-1}(\varnothing) \coloneqq \varnothing$.

3.3.1.1 Lemma 1

*Lemma:*

Given the general definition of the transformation functions and their inverse, for any $q \in \mathbf{R}^k$, then $q \in f(f^{-1}(q))$.

*Proof:*

Given a vector *q*, let

$$A = f^{-1}(q).$$

Note that this general inverse function is not assumed to be a one-to-one fully invertible function. From the generalized definition of the transformative function over set *A*, $f(S) \coloneqq \bigcup_{s \in S} f(s)$, the function is composed of one or more subsets. Given two sets $S_1$ and $S_2$ such that $S_1 \subseteq S_2$, it follows that $f(S_1) \subseteq f(S_2)$. Based upon this result, it follows that for any *a'* that is an element of *A* (i.e. $a' \in A$)

$$f(a') \subseteq f(A).$$

Substituting *a'* for *s* in the definition of the inverse function $f^{-1}(q) \coloneqq \{s \mid q \in f(s)\}$ states that $q \in f(a')$ for some $a' \in A$. Therefore it is known that

$$q \in f(a') \subseteq f(A).$$

And substituting the definition of the set *A*, $A = f^{-1}(q)$, into the previous equation shows the following

$$q \in f(a') \subseteq f(A) = f(f^{-1}(q)) .$$

Therefore

$$q \in f(f^{-1}(q)) .$$

□

3.3.1.2 Lemma 2

*Lemma:*

Given the set $B \subseteq \mathbf{R}^k$, then $B \subseteq f(f^{-1}(B))$.

*Proof:*

To show this by contradiction, the statement is assumed to be not true, such that for some $q \in B$ but $q \notin f(f^{-1}(B))$, which means that it is assumed that $q$ is an element of $B$ but not an element of $f(f^{-1}(B))$.

From the definition of $f^{-1}(B)$ it is clear that for $q \in B$, it can be stated that

$$f^{-1}(q) \subseteq f^{-1}(B) .$$

From the definition of the transformation function *f*(*S*), given two sets $A_1$ and $A_2$ such that $A_1 \subseteq A_2$, it can be stated that $f(A_1) \subseteq f(A_2)$. Then the following is true

$$f(f^{-1}(q)) \subseteq f(f^{-1}(B)) .$$

Given this result and the previous result that $q \in f(f^{-1}(q))$ and that $q \in B$ this shows a contradiction to the statement that $q \in B$ but $q \notin f(f^{-1}(B))$, therefore

$$B \subseteq f(f^{-1}(B)) .$$

□

3.3.1.3 Lemma 3

*Lemma:*

Given the set $S \subseteq \mathbf{R}^k$, then $S \subseteq f^{-1}(f(S))$.

*Proof:*

To show this by contradiction, the statement is assumed to be not true, such that for some $s \in S$ but $s \notin f^{-1}(f(S))$, which means that it is assumed that $s$ is a subset of $S$ but not an element of $f^{-1}(f(S))$.

Given a set of vectors $S$, let

$$T = f(S).$$

From the definition of the pseudo-inverse function $f^{-1}(q) := \{s \mid q \in f(s)\}$, choose some $t$ such that $t \in f(s)$. Then it can be stated that

$$s \in f^{-1}(t).$$

Since $s \in S$, then $t \in T$. Also, since $s \in f^{-1}(t)$ and $t \in f(s)$, then

$$s \in f^{-1}(f(s)).$$

Note that this result is very similar to the result of Lemma 1.

Since $t \in T$, then $f^{-1}(t) \subseteq f^{-1}(T)$ and given that $t \in f(s)$ and $T = f(S)$, then

$$f^{-1}(f(s)) \subseteq f^{-1}(f(S))$$

Therefore

$$s \in f^{-1}(f(s)) \subseteq f^{-1}(f(S)),$$

so

$$s \in f^{-1}(f(S)),$$

which contradicts the assumption that $s \in S$ but $s \notin f^{-1}(f(S))$. Therefore given the set $S \subseteq \mathbf{R}^k$, then

$$S \subseteq f^{-1}(f(S)). \qquad \square$$

### 3.3.2 Simultaneous vs. Sequential Transformations

In order to further understand the transformation of regions in the node-arc network, consider how the order of transforming the regions might affect subsequent transformations. Consider a generic node with an incoming region *X* and an outgoing region *Y*. A simultaneous transformation is defined as conducting a forward transformation of the region *X* and at the same time (prior to updating the resulting region) conducting a pseudo-inverse transformation on the region *Y*. Since one transformed region is not affected by the other transformed region at a given time, *t*, the order of forward or inverse transformation is irrelevant.

A sequential transformation is defined as performing a forward transformation on region *X* and then using the newly altered region *Y* for the inverse transformation to update the *X* region. Note that the order of forward or inverse transformations may have an effect on subsequent transformations in the node-arc network.


Figure 3.3: Transformation of Regions

Consider the simultaneous (in parallel) transformations of regions for the diagrams shown above, which results in the following equations for some time, $t = 0$.

$$Y_P(1) = f(X(0)) \bigcap Y(0)$$

$$X_P(1) = f^{-1}(Y(0)) \bigcap X(0)$$

The regions can also be transformed sequentially (in series) to yield the following results:

$$Y_S(1) = f(X(0)) \bigcap Y(0)$$

$$X_S(1) = f^{-1}(Y_S(1)) \bigcap X(0)$$

These sets of equations appear to yield different results dependent upon the order in which the regions are transformed. Further examination is needed to determine which method is better to transform the regions or if there is a difference. Examination of the sequential regions yields the following results by substitution of the $Y_S(1)$ region:

$$X_S(1) = f^{-1}(Y_s(1)) \bigcap X(0)$$

$$X_S(1) = f^{-1}(f(X(0)) \bigcap Y(0)) \bigcap X(0)$$

$$X_S(1) = f^{-1}(f(X(0))) \bigcap f^{-1}(Y(0)) \bigcap X(0)$$

Recall from Lemma 3 that given the set $S \subseteq \mathbf{R}^k$, then $S \subseteq f^{-1}(f(S))$. Therefore since $X(0) \subseteq \mathbf{R}^k$, then $f^{-1}(f(X(0))) \supseteq X(0)$, which allows $X(0)$ be substituted in the $X(s)$ equation for $f^{-1}(f(X(0)))$, even though $X(0)$ is a subset. This is because the first two terms are intersected with $X(0)$. The substitution of gives the following equation

$$X_S(1) = X(0) \bigcap f^{-1}(Y(0)) \bigcap X(0) = f^{-1}(Y(0)) \bigcap X(0).$$

Recall that $X_P(1) = f^{-1}(Y(0)) \bigcap X(0)$, therefore

$$X_S(1) = X_P(1).$$

Since $X_S(1)$ is equivalent to $X_P(1)$ the transformation order is irrelevant. Therefore, a transformation for iteration $i + 1$ for some generic node $n$ with an incoming arc region $X$ and outgoing arc region $Y$, is defined as

$$Y(i+1) = f_n(X(i)) \bigcap Y(i)$$

$$X(i+1) = f_n^{-1}(Y(i+1)) \bigcap X(i).$$

### 3.3.3 Splitting of Regions

The results of the previous sections apply to incoming sets *X* and outgoing sets *Y*, even though *X* (or *Y*) may be associated with incoming (or respectively outgoing) arcs. The effects of the transformations on the individual incoming or outgoing arcs are now considered when there is more than one arc into or out of the node.

Consider a portion of a larger node network, in which a single node has two incoming arcs with regions *A* and *B* and a single outgoing arc with a single region *C*, as shown below.

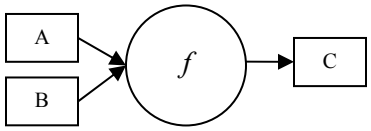


Figure 3.4: Multiple Incoming Regions

The current state of the regions is assumed to be at a particular instant in time that will be considered the initial state and is denoted as *X*(0) for a given region. The forward transformation of the *A* and *B* regions potentially creates two regions from region *C*. The two new regions may or may not be a subset or superset of the other, but there is not sufficient information, at this point in time, to determine the uniqueness of each region so the worst-case is assumed as two unique regions. The forward transformation yields the following regions from the original region *C*. Note that the cycle index number increases as the transformations propagate.

$$C_1(1) = f(A(0)) \bigcap C(0)$$

$$C_2(1) = f(B(0)) \bigcap C(0)$$

The two new regions are now inversely transformed and intersected with the *A* and *B* regions using the sequential transformation method to update the incoming regions. Note that the number of new regions is simply the number of regions being transformed multiplied times the number of regions being intersected.

$$A_1(1) = f^{-1}(C_1(1)) \bigcap A(0)$$

$$A_2(1) = f^{-1}(C_2(1)) \bigcap A(0)$$

$$B_1(1) = f^{-1}(C_1(1)) \bigcap B(0)$$

$$B_2(1) = f^{-1}(C_2(1)) \bigcap B(0)$$

Consider the regions $A_i$. Is it possible to claim that some $q$ exists such that $q \in A_2(1)$ but $q \notin A_1(1)$? The statement that $q \in A_2(1)$ implies that

$$q \in f^{-1}(C_2(1)) \bigcap A(0) = f^{-1}(f(B(0)) \bigcap C(0)) \bigcap A(0)$$
$$= f^{-1}(f(B(0))) \bigcap f^{-1}(C(0)) \bigcap A(0)$$

Therefore $q$ must be an element of the following sets

$$q \in f^{-1}(f(B(0))$$
$$q \in f^{-1}(C(0))$$
$$q \in A(0)$$

Now consider the statement $q \notin A_1(1)$, where $A_1(1)$ is equivalent to

$$A_1(1) = f^{-1}(C_1(1)) \bigcap A(0) = f^{-1}(f(A(0)) \bigcap C(0)) \bigcap A(0)$$
$$= f^{-1}(f(A(0))) \bigcap f^{-1}(C(0)) \bigcap A(0)$$
$$= f^{-1}(C(0)) \bigcap A(0)$$

Therefore one of the following statements must be true if $q \notin A_1(1)$

$$q \notin A(0) \subseteq f^{-1}(f(A(0)))$$
$$q \notin f^{-1}(C(0))$$

Clearly this is a contradiction of the statement that $q \in A_2(1)$ but $q \notin A_1(1)$, because it was previously shown that $q$ is an element of $A(0)$ and $f^{-1}(C(0))$. Therefore

$$A_2(1) \subseteq A_1(1).$$

A similar argument can be used for the $B_1(1)$ and $B_2(1)$ regions. Is it possible for some $q$ to exist such that $q \in B_1(1)$ but $q \notin B_2(1)$? The statement of $q \in B_1(1)$ means that

$$\begin{aligned} q \in B_1(1) &= f^{-1}(C_1(1)) \bigcap B(0) \\ &= f^{-1}(f(A(0)) \bigcap C(0)) \bigcap B(0) \\ &= f^{-1}(f(A(0))) \bigcap f^{-1}(C(0)) \bigcap B(0) \end{aligned}$$

Therefore $q$ is an element of the following sets

$$q \in B(0)$$
$$q \in f^{-1}(C(0))$$
$$q \in f^{-1}(f(A(0)))$$

Now consider the statement that $q \notin B_2(1)$, where

$$\begin{aligned} B_2(1) &= f^{-1}(C_2(1)) \bigcap B(0) \\ &= f^{-1}(f(B(0)) \bigcap C(0)) \bigcap B(0) \\ &= f^{-1}(f(B(0))) \bigcap f^{-1}(C(0)) \bigcap B(0) \\ &= f^{-1}(C(0)) \bigcap B(0) \end{aligned}$$

Therefore one of the following statements must be true if $q \notin B_2(1)$

$$q \notin B(0) \subseteq f^{-1}(f(B(0)))$$
$$q \notin f^{-1}(C(0))$$

Clearly this is a contradiction of the statement that $q \in B_1(1)$ but $q \notin B_2(1)$, because it was previously shown that $q$ is an element of $f^{-1}(C(0))$ and $B(0)$. Also since it was shown that $B(0)$ is a subset of $f^{-1}(B(0))$, $q$ is also an element of $f^{-1}(f(B(0)))$. Therefore

$$B_1(1) \subseteq B_2(1).$$

### 3.3.4 Repeated Region Transformations

Consider the previous example in which the initial regions have been transformed and inversely transformed such that the regions have been defined as $A_1(1)$, $B_2(1)$, $C_1(1)$, and $C_2(1)$. Now consider if the regions were transformed again before any of the regions are changed by adjacent nodes. The forward transformation will yield the following equations for region *C*.

$$\begin{aligned}
C_1(2) &= f(A_1(1)) \bigcap C_1(1) \\
&= f(f^{-1}(C_1(1)) \bigcap A(0)) \bigcap C_1(1) \\
&= f(f^{-1}(C_1(1))) \bigcap f(A(0)) \bigcap C_1(1) \\
&= f(A(0)) \bigcap C_1(1) \\
&= f(A(0)) \bigcap f(A(0)) \bigcap C(0) \\
&= f(A(0)) \bigcap C(0) \\
&= C_1(1)
\end{aligned}$$

$$\begin{aligned}
C_2(2) &= f(B_2(1)) \bigcap C_2(1) \\
&= f(f^{-1}(C_2(1)) \bigcap B(0)) \bigcap C_2(1) \\
&= f(f^{-1}(C_2(1))) \bigcap f(B(0)) \bigcap C_2(1) \\
&= f(B(0)) \bigcap C_2(1) \\
&= f(B(0)) \bigcap f(B(0)) \bigcap C(0) \\
&= f(B(0)) \bigcap C(0) \\
&= C_2(1)
\end{aligned}$$

This shows that the repeated transformation of the given regions has no affect on the regions because the new region is equivalent to the previous region. Consider the following cross transformations

$$
\begin{aligned}
C_{12}(2) &= f(A_1(1)) \bigcap C_2(1) \\
&= f(f^{-1}(C_1(1)) \bigcap A(0)) \bigcap f(B(0)) \bigcap C(0) \\
&= f(f^{-1}(C_1(1))) \bigcap f(A(0)) \bigcap f(B(0)) \bigcap C(0) \\
&= f(f^{-1}(f(A(0)) \bigcap C(0))) \bigcap f(A(0)) \bigcap f(B(0)) \bigcap C(0)) \\
&= f(f^{-1}(f(A(0)))) \bigcap f(f^{-1}(C(0))) \bigcap f(A(0)) \bigcap f(B(0)) \bigcap C(0) \\
&= f(A(0)) \bigcap f(B(0)) \bigcap C(0)
\end{aligned}
$$

$$
\begin{aligned}
C_{21}(2) &= f(B_2(1)) \bigcap C_1(1) \\
&= f(f^{-1}(C_2(1)) \bigcap B(0)) \bigcap f(A(0)) \bigcap C(0) \\
&= f(f^{-1}(C_2(1))) \bigcap f(B(0)) \bigcap f(A(0)) \bigcap C(0) \\
&= f(f^{-1}(f(B(0)) \bigcap C(0))) \bigcap f(B(0)) \bigcap f(A(0)) \bigcap C(0)) \\
&= f(f^{-1}(f(B(0)))) \bigcap f(f^{-1}(C(0))) \bigcap f(B(0)) \bigcap f(A(0)) \bigcap C(0) \\
&= f(A(0)) \bigcap f(B(0)) \bigcap C(0)
\end{aligned}
$$

Note that the cross transformation of the regions result in creating subsets of the original transformed regions. Both of the sets $C_{12}(1)$ and $C_{21}(1)$ are a subset of $C_1(1)$ or $C_2(1)$.

Now consider the repeated inverse transformations of the *C* regions when the *C* regions have not been changed by the adjacent nodes. The inverse transformation will yield the following equations for the *A* and *B* regions:

$$
\begin{aligned}
A_1(2) &= f^{-1}(C_1(1)) \bigcap A_1(1) \\
&= f^{-1}(C_1(1)) \bigcap f^{-1}(C_1(1)) \bigcap A(0) \\
&= A_1(1)
\end{aligned}
$$

$$
\begin{aligned}
A_2(2) &= f^{-1}(C_2(1)) \bigcap A_1(1) \\
&= f^{-1}(C_2(1)) \bigcap f^{-1}(C_1(1)) \bigcap A(0) \\
&\subseteq A_1(1)
\end{aligned}
$$

$$B_1(2) = f^{-1}(C_1(1)) \bigcap B_2(1)$$
$$= f^{-1}(C_1(1)) \bigcap f^{-1}(C_2(1)) \bigcap B(0)$$
$$\subseteq B_2(1)$$

$$B_2(2) = f^{-1}(C_2(1)) \bigcap B_2(1)$$
$$= f^{-1}(C_2(1)) \bigcap f^{-1}(C_2(1)) \bigcap B(0)$$
$$= B_2(1)$$

The results for the sets $C_1(2)$, $C_2(2)$, $A_1(2)$, and $B_2(2)$, in this example illustrate that repeated transformations, whether forward or backward, of unaltered regions has no affect on the other regions at the node. Therefore the following statement can be made:

> The incoming and outgoing regions for a given node will not be altered by any subsequent transformations or inverse transformations unless one or more regions have been altered by the transformation or inverse transformation at an adjacent node.

Note that the node-arc network is used in this research to determine feasible buffer levels (represented by a region) at a given node which represents idle time, setup, or production of a given product. After all the nodes in the system have been cycled through and the regions are no longer altered by the transformation functions, the analysis of the node-arc network is complete.

### 3.3.5 Transformation of Single Incoming New Region

Consider the introduction of an altered region $A_1(2)$, which is a subset of $A_1(1)$, to the previous example at the given time at which the other regions are defined as $B_2(1)$, $C_1(1)$, and $C_2(1)$. Given the following incoming regions

$$A_1(2) \subseteq A_1(1)$$
$$B_2(2) = B_2(1)$$

which are shown in the diagram below.

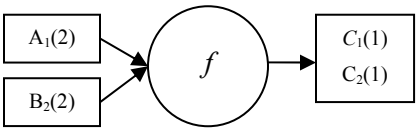

Figure 3.5: Multiple Incoming Arcs

The forward transformation of the regions $A_1(2)$ and $B_2(2)$ will yield the following

$$\begin{aligned} C_1(2) &= f(A_1(2)) \bigcap C_1(1) \\ &= f(A_1(2)) \bigcap f(A(0)) \bigcap C(0) \\ &= f(A_1(2)) \bigcap C(0) \end{aligned}$$

$$\begin{aligned} C_2(2) &= f(B_2(2)) \bigcap C_2(1) \\ &= f(B_2(1)) \bigcap C_2(1) \\ &= f(f^{-1}(C_2(1)) \bigcap B(0)) \bigcap C_2(1) \\ &= f(f^{-1}(C_2(1))) \bigcap f(B(0)) \bigcap C_2(1) \\ &= f(B(0)) \bigcap C_2(1) \\ &= f(B(0)) \bigcap f(B(0)) \bigcap C(0) \\ &= f(B(0)) \bigcap C(0) \\ &= C_2(1) \end{aligned}$$

The cross transformation of the region $A_1(2)$ is as follows

$$\begin{aligned} C_{12}(2) &= f(A_1(2)) \bigcap C_2(1) \\ &= f(A_1(2)) \bigcap f(B(0)) \bigcap C(0) \end{aligned}$$

which leads to both

$$C_{12}(2) \subseteq f(A_1(2)) \bigcap C(0) = C_1(2)$$
$$C_{12}(2) \subseteq f(B(0)) \bigcap C(0) = C_2(2)$$

The cross transformation of the region $B_2(2)$ is as follows

$$
\begin{aligned}
C_{21}(2) &= f(B_2(2)) \bigcap C_1(1) \\
&= f(B_2(1)) \bigcap C_1(1) \\
&= f(f^{-1}(C_2(1)) \bigcap B(0)) \bigcap C_1(1) \\
&= f(f^{-1}(C_2(1))) \bigcap f(B(0)) \bigcap C_1(1) \\
&= f(f^{-1}(f(B(0)) \bigcap C(0))) \bigcap f(B(0)) \bigcap C_1(1) \\
&= f(f^{-1}(f(B(0)) \bigcap C(0))) \bigcap f(B(0)) \bigcap f(A(0)) \bigcap C(0) \\
&= f(f^{-1}(f(B(0)))) \bigcap f(f^{-1}(C(0))) \bigcap f(B(0)) \bigcap f(A(0)) \bigcap C(0) \\
&= f(B(0)) \bigcap f(A(0)) \bigcap C(0) \subseteq C_2(2)
\end{aligned}
$$

Therefore the cross transformation regions can be disregarded because they will be contained within the $C_1(2)$ or $C_2(2)$ sets or the cross transformation regions will be empty. The forward transformations of the regions $A_1(2)$ and $B_2(2)$ have only altered the region $C_1(2)$ since the region $C_2(2)$ is equivalent to $C_2(1)$.

Now consider the inverse transformation of the regions $C_1(2)$ and $C_2(2)$

$$
\begin{aligned}
A_1(3) &= f^{-1}(C_1(2)) \bigcap A_1(2) \\
&= f^{-1}(f(A_1(2)) \bigcap C(0)) \bigcap A_1(2) \\
&= f^{-1}(f(A_1(2))) \bigcap f^{-1}(C(0)) \bigcap A_1(2) \\
&= f^{-1}(C(0)) \bigcap A_1(2)
\end{aligned}
$$

$$
\begin{aligned}
A_2(3) &= f^{-1}(C_2(2)) \bigcap A_1(1) \\
&= f^{-1}(f(B(0)) \bigcap C(0)) \bigcap A_1(1) \\
&= f^{-1}(f(B(0))) \bigcap f^{-1}(C(0)) \bigcap A_1(1) \subseteq A_1(3)
\end{aligned}
$$

$$
\begin{aligned}
B_1(3) &= f^{-1}(C_1(2)) \bigcap B_2(2) \\
&= f^{-1}(f(A_1(2)) \bigcap C(0)) \bigcap f^{-1}(C_2(1)) \bigcap B(0) \\
&= f^{-1}(f(A_1(2))) \bigcap f^{-1}(C(0)) \bigcap f^{-1}(f(B(0)) \bigcap C(0)) \bigcap B(0) \\
&= f^{-1}(f(A_1(2))) \bigcap f^{-1}(C(0)) \bigcap f^{-1}(f(B(0))) \bigcap f^{-1}(C(0)) \bigcap B(0) \\
&= f^{-1}(f(A_1(2))) \bigcap f^{-1}(C(0)) \bigcap B(0)
\end{aligned}
$$

Also, $B_1(3) = f^{-1}(C_1(2)) \bigcap B_2(2) \subseteq B_2(2) = B_2(1)$

$$
\begin{aligned}
B_2(3) &= f^{-1}(C_2(2)) \bigcap B_2(2) \\
&= f^{-1}(C_2(1)) \bigcap B_2(2) \\
&= f^{-1}(C_2(1)) \bigcap f^{-1}(C_2(1)) \bigcap B(0) \\
&= f^{-1}(C_2(1)) \bigcap B(0) \\
&= f^{-1}(f(B(0)) \bigcap C(0)) \bigcap B(0) \\
&= f^{-1}(f(B(0))) \bigcap f^{-1}(C(0)) \bigcap B(0) \\
&= f^{-1}(C(0)) \bigcap B(0) \\
&= B_2(1)
\end{aligned}
$$

The results from the inverse transformation are similar to the results from the forward transformation in that only one region has been altered, $A_1(3)$. Note that the cross-inverse transformed regions $A_2(3)$ and $B_1(3)$ are simply subsets (or empty sets) of the regions $A_1(3)$ or $B_2(3)$.

### *3.4 Settled Network*

Consider the regions $X$ and $Y$ at node $n$, for some arc-node network and given the previous definition of the transformation function and inverse function for iteration $i$ + 1, as $Y(i+1) = f_n(X(i)) \bigcap Y(i)$ and $X(i+1) = f_n^{-1}(Y(i+1)) \bigcap X(i)$. The intersection of the transformed region and the original region causes the new region to always be equivalent or less than the original region and the new region can never be larger than the original region. Therefore the following is always true for any set of regions $X$ and $Y$

$$Y(i+1) \subseteq Y(i)$$
$$X(i+1) \subseteq X(i)\,.$$

Therefore as the iterations increase, a region must settle to some set, where the set is either non-empty set or an empty set. A network with $N$ nodes and where each node has an incoming region of $X_n$ and an outgoing region of $Y_n$ is settled at iteration $i$, when for all nodes $n$, $n \in N$, the following occurs

$$X_n(i+1) = X_n(i)$$
$$Y_n(i+1) = Y_n(i)\,.$$

3.4.1.1 Definition of a Settled Network

This leads to the following definition for a settled arc-node network:

> A given network characterized by $\eta = (N, A, f(\bullet), Z(\bullet))$, is *settled* when the following equations are true for every node $n$, where $n \in N$.

$$X_n = f^{-1}(Y_n) \bigcap X_n$$
$$Y_n = f(X_n) \bigcap Y_n$$

## *3.5 Stable Network Trajectory*

The stability of an arc-node network occurs for a settled network with a non-empty region at any node. Stability means that a trajectory exists that can propagate from non-empty set to non-empty set through the network an infinite number of times while never entering an empty set. At any node on the trajectory, the trajectory must start at some element within a non-empty incoming region and be transformed to some element within a non-empty outgoing region. The trajectory is not considered to be stable if at some node the trajectory is transformed to an empty set or the transformed point is not contained within the non-empty outgoing region.

A stable trajectory will exist for a settled arc-node network characterized by $\eta = (N, A, f(\bullet), Z(\bullet))$, if there exists some node $n$ with a settled and non-empty region.

#### 3.5.1.1 Lemma 4

*Lemma:*

For any node *n* in a *settled* arc-node network with $X_n \neq \varnothing$, then $Y_n \neq \varnothing$.

*Proof:*

This Lemma can be proved by contradiction, suppose that $Y_n = \varnothing$. A settled network was defined previously as $X_n = f^{-1}(Y_n) \bigcap X_n$ and that $f^{-1}(\varnothing) = \varnothing$. Therefore $X_n = \varnothing \bigcap X_n = \varnothing$, since $X_n = \varnothing$, this is a contradiction.

□

#### 3.5.1.2 Lemma 5

*Lemma:*

For any node *n* in a *settled* arc-node network with $Y_n \neq \varnothing$, then $X_n \neq \varnothing$.

*Proof:*

This Lemma can be proved in a similar manner to Lemma 4. A settled network was defined previously as $Y_n = f(X_n) \bigcap Y_n$ and that $f(\varnothing) := \varnothing$. Therefore if $X_n = \varnothing$, then $Y_n = \varnothing \bigcap Y_n = \varnothing$, which is a contradiction.

□

#### 3.5.1.3 Definition of a Stable Trajectory

Lemma 4 and Lemma 5 and the definition of a settled network lead to the following definition:

A stable trajectory of a *settled* arc-node network, characterized by $\eta = (N, A, f(\bullet), Z(\bullet))$, is an endless path $a_{n_i,n_{i+1}}, a_{n_{i+1},n_{i+2}}, \ldots$ through the network (where nodes and arcs can be revisited an infinite number of times) and the corresponding sequence of values $z_i, z_{i+1}, z_{i+2}, \ldots$, such that for each $i > 1$, $z_{i+1} \in Z(a_{n_i,n_{i+1}})$ and $z_{i+1} \in f_{n_{i+1}}(z_i)$.

#### 3.5.1.4 Theorem 1

*Theorem:*

Given a *settled* arc-node network, if there exists some arc $a_{n_i,n_{i+1}} \in A$ with $Z(a_{n_i,n_{i+1}}) \neq \varnothing$, then there exists a stable trajectory for the network.

*Proof:*

This result follows directly from Lemma 4 and Lemma 5. Consider some arc with $Z(a_{n_i,n_{i+1}}) \neq \varnothing$ for a settled network. Let $X_{n_i} = Z(a_{n_i,n_{i+1}})$, since the arc is unidirectional from $n_i$ toward the next node $n_{i+1}$ in the trajectory sequence. Given that $Z(a) \neq \varnothing$, then $X_n \neq \varnothing$, therefore from, $Y_n \neq \varnothing$. Consider that $Y_{n_i} = Z(a_{n_{i+1},n_{i+2}})$ is equivalent to $Y_{n_i} = X_{n_{i+1}} = Z(a_{n_{i+1},n_{i+2}})$. Therefore the outgoing region *Y* for a given node is the incoming region *X* for the next node in the path stable trajectory.

□

3.5.1.5 Propagation of a Stable Trajectory

The results of Lemma 4 and Lemma 5 prove that any element of a non-empty incoming set $X_n$ will map to some element of a non-empty outgoing set $Y_n$ for a given node *n* in a settled network. Recall that a settled network is one in which the regions on all arcs contained within $Arc_{in}(n)$ and $Arc_{out}(n)$ are unchanged after being intersected with some region transformed by the node. Thus for any node *n*,

$$Y_n = f_n(X_n) \bigcap Y_n$$

and

$$X_n = f_n^{-1}(Y_n) \bigcap X_n .$$

Therefore any settled arc-node network with a node that has a non-empty set on any arc will have a stable trajectory that will cycle through the network. This will occur because node $n_i$ is connected to the rest of the network with directed arcs contained in $Arc_{in}(n_i)$ and $Arc_{out}(n_i)$, in which the outgoing set $Y_{n_i}$ for node $n_i$ is equivalent to the incoming set $X_{n_{i+1}}$ for node $n_{i+1}$, given that $a(n_i,n_{i+1})$ is contained within $Arc_{out}(n_i)$. This means that any element of $X_{n_i}$ will map to some element in the outgoing set $Y_{n_i}$, which is the same as an element in $X_{n_{i+1}}$. The element in $X_{n_{i+1}}$ will map to some element in $Y_{n_{i+1}}$, which is the same as an element in $X_{n_{i+2}}$, given that $a(n_{i+1},n_{i+2})$ is contained within $Arc_{out}(n_{i+1})$. This mapping from incoming region to outgoing region will continue an infinite number of times and each mapping will always map from a non-empty incoming set to a non-empty outgoing set.

# 4 Stability Algorithm

## *4.1 Introduction*

A multi-product system in which buffers are replenished completely, often referred to as a switched arrival system, is a difficult system to analyze for stability. This type of system with more than two or three products quickly becomes difficult to manage by hand and practically impossible to visualize the interactions between the products. The purpose of this stability algorithm is to aid in the discussion and understanding of the multi-product system and the role of different parameters upon the existence of stability. Chase, Serrano, and Ramadge [5] show that this type of a system is chaotic in nature when they examined a three product system in which the server could move instantaneously to replenish a buffer that had reached the triggering threshold level. Horn and Ramadge [75] examined the effects of imposing upper and lower buffer thresholds upon the previously mentioned switched arrival system. The results show that lower limits had no affect on the system dynamics while upper limits can switch the system from chaos to a stable periodic orbit. Ushio [76] has conducted a significant amount of research into methods of controlling chaos by limiting the refill time to force the server to switch earlier to the next buffer.

The production system being examined in this research is more complicated than previous work and is a more accurate representation of manufacturing systems. The system is a multi-product system that may or may not experience idle time between replenishment of a product and the setup of the next product. A setup time may or may not exist before the replenishment of each buffer, depending upon how the system is defined. If present the setup may be sequence dependent, meaning that setup time may vary based upon the previous product. If a product is repeated, it is assumed that whether or not idle time exists between the two replenishments, a setup will occur prior to replenishment. It is also assumed that the product buffer is always completely replenished and never interrupted by another product. The production system and parameters are assumed to be constant and contain no variability, such as breakdowns, maintenance, or other interruptions.

The algorithm cycles through a node network of the system until all nodes have incoming and outgoing regions that are equivalent when transformed by the node, at which point the network is *settled*, see Node Network Analysis Chapter, Section 4.1.1 for a formal definition. When the regions no longer change, the system is considered to be balanced and stable regions will have

been found by the algorithm. Initially all regions are set with a minimum value of zero and a maximum value of a full buffer level for each product. As the node network is cycled through, the incoming regions are transformed forward and the outgoing regions are transformed backward for each flagged node until the transformed regions are equivalent to the regions from the previous cycle.

## *4.2 Basic System Model*

The system that is being evaluated by the stability algorithm is a network of nodes that are connected together by arcs, as described in the previous chapter. Each arc contains one or more product regions (the rectangles in the diagram below), currently in the form of a matrix, that can be operated upon by the upstream or downstream node. Each node is host to a set of transformation functions and/or constraints depending upon which node type is defined for the given node.

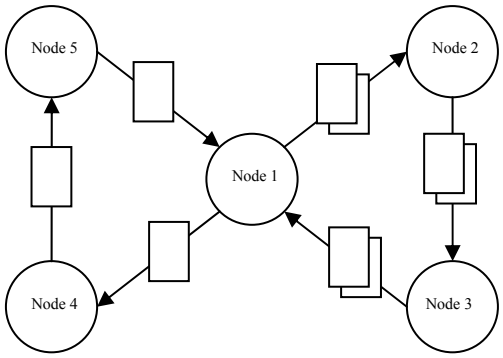


Figure 4.1: Basic Model Arc-Node Network

### 4.2.1 Product Inventory Data Handling

The product buffer information is associated with the arcs in the arc-node network system. The product inventory is a region represented by the minimum buffer level and maximum buffer level for each product in the system. As in the previous chapter, the system contains $k$ products and an arbitrary inventory state is a vector of length $k$, indicating the inventory levels of the $k$ products. In this chapter arbitrary sets of inventory states are not considered, rather sets of inventory states represented by the union of sets defined by the minimum and maximum values of each product are considered. The inventory state sets can be geometrically represented by a hyperrectangle of dimensions $k - 1$, such that a two product system has regions that are represented by lines, a three product system has regions that are represented by rectangles, a four product system has regions represented by boxes, etc.

Any arc $a$ in the system may have one or more product buffer regions associated with the arc, or an empty set region. The product buffer region(s) on arc $a$ is one or more $k \times 2$ matrices, where $k$ is the number of products, stored in $\mathcal{Z}(a)$ as a union over all regions $Z_J(a)$. Multiple regions on a given arc are stored in the algorithm as a matrix for each region, where $j$ is the index of a region on a given arc and $J$ is the maximum number of regions on a given arc.

$$Z_1(a) = \begin{bmatrix} z_{\min,1}(1) & z_{\max,1}(1) \\ \vdots & \vdots \\ z_{\min,1}(k) & z_{\max,1}(k) \end{bmatrix}, \text{ where } j = 1$$

$$\mathcal{Z}(a) = \bigcup_{j=1}^{J} \begin{bmatrix} z_{\min,j}(1) & z_{\max,j}(1) \\ \vdots & \vdots \\ z_{\min,j}(k) & z_{\max,j}(k) \end{bmatrix}$$

The first column is the minimum product value and the second column is the maximum product value for the given arc, $a$. Initially, all product regions have a minimum value of zero and a maximum value of the full buffer, $U(k)$.

For clarity, the remainder of this chapter will use a localized notation centered about a given node $n$, where a set of points forming a hyperrectangle on an incoming arc is denoted as $X$ and a hyperrectangular set of points on an outgoing arc is denoted as $Y$. The set of directed arcs leading to node $n$ are denoted as $Arc_{in}(n)$ and the set of directed arcs originating from $n$ are denoted as $Arc_{out}(n)$ (i.e. $Arc_{in}(n) = \{a(n', n) \in A \text{ for some } n' \in N\}$ and $Arc_{out}(n) = \{a(n, n') \in A \text{ for some } n' \in N\}$). A set of hyperrectangle inventory regions entering a node $n$ is $\mathcal{X}(a_{in})$, and a set of inventory vectors leaving node $n$ is $\mathcal{Y}(a_{out})$. Note that the script font indicates a set of regions, of which each set is hyperrectangular.

The localized $\mathcal{X}$ represents the set of all regions associated with an arc $a_{in}$ contained in $Arc_{in}(n)$, such that

$$\mathcal{X}(a_{in}) = \bigcup_{j=1}^{J} \left( Z(a_{in}) \right).$$

The localized $\mathcal{Y}$ similarly represents the set of all regions associated with an arc $a_{out}$ contained within $Arc_{out}(n)$, such that

$$\mathcal{Y}(a_{out}) = \bigcup_{j=1}^{J} \left( Z(a_{out}) \right).$$

### 4.2.2 Basic Arc Type

An arc is simply a means of conveying and directing the status of each product buffer between the nodes in the network. The regions are unchanged by the arc and are only transformed by a node. An arc also allows the direction of flow to be defined between each node because the arcs used for this algorithm are unidirectional. There must always be one or more arc(s) into a given node and one or more arcs out of a given node.

### 4.2.3 Basic Node Types

In the most general sense, the nodes define an increase or decrease, or combination thereof, for the products of the system. The nodes may also define exit conditions based on the time spent at the node or incoming or outgoing product buffer levels. The node network considered in this chapter is narrowed to only consider three types of nodes to represent a production system: idle nodes, setup nodes, and refill nodes.

#### 4.2.3.1 Idle Node

An idle node is entered by the system when all the product buffers are between being completely full and reaching the lower threshold signal level, which signals the system to replenish the buffer of a given product. All products are consumed based on the usage rate of each product while the system is idle. The time spent at an idle node is dependent upon the buffer levels entering the node, lower threshold levels, and usage rates of the products.

#### 4.2.3.2 Setup Node

A setup node is entered by the system prior to refilling a product and a setup always occurs before the replenishment of each product. This type of node is entered only when a product is at or below the lower threshold signal level. The time spent at a setup node is defined by the user. Setup time can vary from product to product as well as possibly varying for a single product based upon what product was previously refilled, i.e. sequence dependent setups. All products are consumed based on the usage rate of each product while the system is undergoing a setup.

#### 4.2.3.3 Refill Node

A refill node is entered only after a setup node for the same product. The buffer level for the product that is being refilled can be empty just as the product enters the refill node without crashing the system and a refill node will always completely refill the buffer. All products that are not being refilled are consumed based on the usage rate of each product while the system is at the refill node. The time spent at a refill node is based upon the buffer level when entering the node, the full buffer level, the usage rate, and production rate for the product being refilled.

## *4.3 Node Transformation Functions*

The production system is represented by a system of nodes connected by arcs, similar to the diagram below. The nodes represent where the system is transforming the product buffer regions and the arcs are simply for connectivity. The regions that the algorithm attempts to determine are for the conditions as an arc is entering a node or exiting a node.

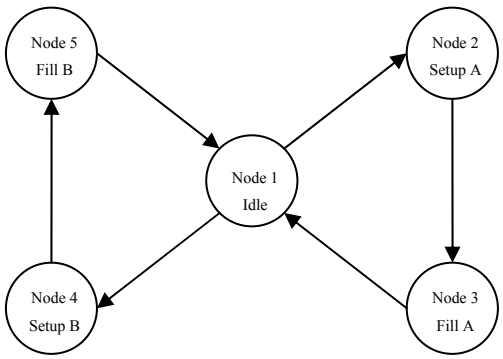


Figure 4.2: Two-Product Network

At each node a function $f(x)$ or pseudo-inverse function $f^{-1}(x)$ is applied to transform the region(s) on one side of the node to create a new region(s) on the other side of the node, see diagram below.

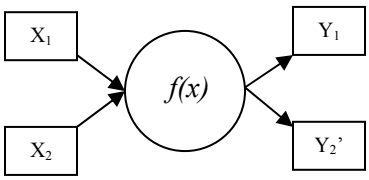


Figure 4.4: Transforming Incoming Regions

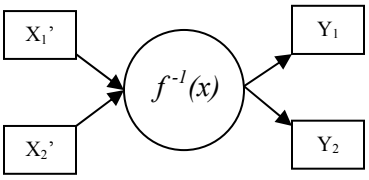


Figure 4.3: Transforming Outgoing Regions

The transformation function for a node is dependent upon the type of node, whether it is a setup node, idle node, or fill node. The system parameters such as setup time, usage rates, replenishment rates, maximum buffer levels, etc. are used in calculating the transformation function. An idle node consumes each product based upon its usage rate until a lower threshold is reached. A setup node also consumes each product based on its usage rate for a time period defined by the setup time from the previous product refilled to the current product preparing to be filled. A production node is the third and final type of node in which the buffer of one product is completely refilled while all other products are consumed at each product's usage rate for a minimum and maximum time period. The minimum production time is the time required to refill the product from the maximum incoming buffer level to a full buffer, while the maximum production time is the time required to refill the buffer completely from the lowest incoming buffer level for the product.

The lowest a buffer can be is zero, although system parameters can cause the lower limit to be higher than zero at any node. The system is assumed to crash if a buffer reaches zero for any product at any node other than the refill node. The product that is being replenished may reach zero just as it enters the production node and the system will not crash.

### 4.3.1 Forward Transformation Functions

#### 4.3.1.1 Idle Node

The idle node consumes all products until a product reaches the lower threshold level, $l(TP)$, which signals the system that a product needs to be replenished. Each arc leaving the idle node represents a product reaching the triggering threshold to be replenished. The transformation functions of the idle node are dependent upon which product has reached the lower threshold and is ready to be replenished. The following discussion is generalized for a given outgoing arc that has a particular product associated with it that has reached the lower threshold.

For outgoing regions there are two sets of constraints to be applied to the regions. The product $k$ that is advancing to the setup node and will be the next product to be refilled is referred to as the triggering product, $TP$. The triggering product will only exit an idle node when the buffer reaches the lower threshold level and cannot be lower than the lower threshold limit. Therefore both the maximum and minimum values for the trigger product region, $TP$, are set at the lower threshold

level, *l(TP)*. There are two possible times for the trigger product to reach the lower threshold value and will be referred to as minimum idle time, $IT_{min}$ and maximum idle time, $IT_{max}$. The minimum idle time is the time required to consume the trigger product from the minimum incoming level to the lower threshold level. The maximum idle time is the time required to consume the trigger product from the maximum incoming level to the lower threshold level. The idle time equations are as follows:

$$IT_{max} = (TP_{max,incoming} - l(TP)\ )/\ rho(TP)$$

$$IT_{min} = (TP_{min,incoming} - l(TP)\ )/\ rho(TP)$$

where *rho*(*TP*) is the usage rate of the trigger product, *TP*. The products which are not going to be the next product refilled are referred to as non-trigger products, *NTP*(*k*). It is assumed that no product can be consumed below the lower threshold level *l*(*k*) in an idle node and therefore both non-trigger products are constrained such that the values cannot cross the lower threshold. If a product crosses the product's lower threshold then the non-trigger product becomes the trigger product. The two constraints for the non-trigger products limits the outgoing region levels from crossing the lower threshold value for the given product *k*.

Transformations for Outgoing Regions for Idle Node:

$$TP_{min,outgoing} = TP_{max,outgoing} = l(TP)$$

$$NTP_{min,outgoing}(k) \geq l(k)$$

$$NTP_{max,outgoing}(k) \geq l(k)$$

$$NTP_{min,outgoing}(k) = minimum[\ NTP_{min,incoming}(k),\ NTP_{max,incoming}(k) - (IT_{max} * rho(k))]$$

$$NTP_{max,outgoing}(k) = NTP_{max,incoming}(k) - (IT_{min} * rho(k))$$

#### 4.3.1.2 Setup Node

The setup node transformation equations are very straightforward for all products and indiscriminant of the trigger or non-trigger products. The equation simply subtracts the products consumed from the incoming region value, based on the setup time of the trigger product, *node(i).time_delta*, and the usage rate of the given product, *rho(k)*.

Transformations for Outgoing Regions for Setup Node:

$$Product_{min,outgoing}(k) = Product_{min,incoming}(k) - (node(i).time\_delta * rho(k))$$

$$Product_{max,outgoing}(k) = Product_{max,incoming}(k) - (node(i).time\_delta * rho(k))$$

#### 4.3.1.3 Refill Node

The refill node replenishes the buffer of the trigger product completely and is never interrupted by the triggering of another product. The transformation equations for the trigger product are solely constraints for the outgoing region of the trigger product. The buffer is filled to the maximum buffer level of the trigger product, *U(TP)*. The refill node transformation equations are similar to the setup node equations for non-trigger products, although a minimum and maximum *time_ delta* is calculated based on the time to replenish the trigger product and is used to determine the amount of non-trigger products consumed at a refill node. The maximum time to replenish the trigger product is calculated as the difference between a full buffer and the minimum incoming trigger product value divided by the production rate minus the usage rate. The minimum replenishment time is calculated using the maximum incoming trigger product value. The time equations are as follows:

$$node(i).time\_delta\_max = (U(TP) - TP_{min,incoming})/(PR(TP) - rho(TP))$$

$$node(i).time\_delta\_min = (U(TP) - TP_{max,incoming})/(PR(TP) - rho(TP))$$

The minimum outgoing region value for the non-trigger product assumes that the maximum time delta occurs during the refilling of the trigger product. While the maximum outgoing region value for the non-trigger product is calculated using the minimum time delta for the refill node.

Transformations for Outgoing Regions for Refill Node:

$$TP_{min,outgoing} = TP_{max,outgoing} = U(TP)$$

$$NTP_{min,outgoing}(k) = NTP_{min,incoming}(k) - (node(i).time\_delta\_max * rho(k))$$

$$NTP_{max,outgoing}(k) = NTP_{max,incoming}(k) - (node(i).time\_delta\_min * rho(k))$$

### 4.3.2 Pseudo-Inverse Transformation Functions

#### 4.3.2.1 Idle Node

The inverse transformation equations for an idle node are constraints based on the parameters of the system. The incoming arc regions will be transformed differently depending on if a product is the trigger product or not. The trigger product for the idle node is the same trigger product that would be used for a forward transformation function, meaning the trigger product is not the product that has just been refilled but is the product that has reached the lower threshold and is the next to be refilled. The trigger product is constrained to a minimum value of the lower threshold level, *l(TP)*, and a maximum value is unchanged. The minimum and maximum time required for the trigger product to reach the lower threshold is calculated with the same equations as for the forward transformation of the idle node.

A product can never enter an idle node with a region any higher than a full buffer and can never leave with a region lower that the lower threshold. The minimum value of the non-trigger product is initially constrained to be equal to the lower threshold level during the initialization of the algorithm. Subsequently, the minimum incoming value cannot be lower than the minimum outgoing value, if this occurs, it would require that products are not consumed but rather replenished at the idle node. The minimum incoming inventory level is calculated by adding the maximum amount of inventory consumed at the idle node to the minimum outgoing inventory level of the given non-trigger product.

The maximum incoming value of the non-trigger product cannot be less than the maximum outgoing value because it would require replenishment of the product to occur at the idle node. The maximum value is calculated by adding the minimum amount of inventory consumed at the

idle node to the maximum outgoing inventory level of the given non-trigger product. An equivalent incoming and outgoing value (either maximum or minimum) requires the trigger product to reach the lower threshold just as the idle node is entered.

Inverse Transformations for Incoming Regions for Idle Node:

$$Product_{min,incoming}(k) = l(k)$$

$$NTP_{min,\ incoming}(k) = minimum[\ NTP_{min,outgoing}(k),\ NTP_{min,outgoing}(k) + (IT_{max} * rho(k))]$$

$$NTP_{max,\ incoming}(k) = maximum[\ NTP_{max,outgoing}(k), NTP_{max,outgoing}(k) + (IT_{min} * rho(k))]$$

#### 4.3.2.2 Setup Node

The inverse node transformation equations for the Setup node are the same for both the trigger and non-trigger products. The equation simply adds the products consumed during the setup to the outgoing region value, based on the setup time of the trigger product, *node(i).time_delta*, and the usage rate of the given product, *rho(k)*. The maximum value of the trigger product is constrained to the lower threshold level, *l(k)*.

Inverse Transformations for Incoming Regions for Setup Node:

$$Product_{min,\ incoming}(k) = Product_{min,\ outgoing}(k) + (node(i).time\_delta * rho(k))$$

$$Product_{max,\ incoming}(k) = Product_{max,\ outgoing}(k) + (node(i).time\_delta * rho(k))$$

$$TP_{max,incoming} = l(TP),\ if > l(TP)$$

#### 4.3.2.3 Refill Node

The inverse transformation equations at the refill node have no effect on the incoming values of the trigger product. This is because the replenishment times are calculated using the incoming values of the trigger product and the full buffer level. Inversely transforming the outgoing region

of the trigger product by subtracting the amount replenished at the refill node will yield the original values of the trigger product.

The non-trigger products are inversely transformed similarly to the forward transformation equations for the refill node. The minimum incoming region is calculated by adding the minimum node time multiplied by the usage rate of the product to the minimum outgoing limit. The maximum incoming region value is calculated by adding the maximum outgoing region value to the maximum node time multiplied by the usage rate of the product. The minimum and maximum replenishment times are calculated using the same equations that were used for the forward transformation of the refill node.

Inverse Transformations for Incoming Regions for Refill Node:

$$NTP_{min,incoming}(k) = NTP_{min,outgoing}(k) + ( node(i).time\_delta\_min * rho(k))$$

$$NTP_{max,incoming}(k) = NTP_{max,outgoing}(k) + (node(i).time\_delta\_max * rho(k))$$

The time equations are as the same as for the forward transformation of the refill node. The equations are as follows:

$$node(i).time\_delta\_max = (U(TP) - TP_{min,incoming})/(PR(TP) - rho(TP))$$

$$node(i).time\_delta\_min = (U(TP) - TP_{max,incoming})/ (PR(TP) - rho(TP))$$

### *4.4 Intersection and Merging of Transformed Regions*

At each node there exists one or more incoming arcs and one or more outgoing arcs and each arc will have one or more regions of product values. An example of a two-product system is shown in the diagram below. The node will perform a transformation on incoming regions to create new outgoing regions or perform an inverse transformation on outgoing regions to create new incoming regions. The newly transformed region(s) is then intersected with the appropriate original region and then the newly intersected regions are merged together.

After a sufficient number of cycles through the network the regions will be settled which occurs when a transformed region is equivalent to the existing region at the node. Note that empty sets can occur in a settled network. A stable trajectory will exist for the system when a non-empty set exists for any region in the *settled* network; see Node Network Analysis Chapter, Section 5 for a thorough discussion and proof of this statement.

Initially the product value regions are as large as possible to include all possible stable regions for each node. The minimum region value for product $k$ is equivalent to an empty buffer and the maximum region value is equivalent to a full buffer, $U(k)$. As a node is evaluated the region may remain unchanged, split into multiple regions, or the region may shrink as it approaches the final stable region. A region is never allowed to expand due to the intersection with the existing region.

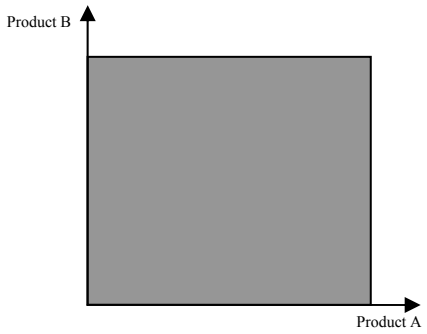


Figure 4.5: Initial Buffer Region

### 4.4.1 Intersection and Union Example

Consider a node that has one incoming arc with an initial product value region $x_1$ and two outgoing arcs with initial product value regions of $Y_1$ and $Y_2$.

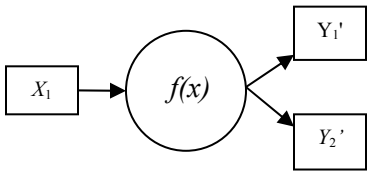


Figure 4.6: Transforming Incoming Regions

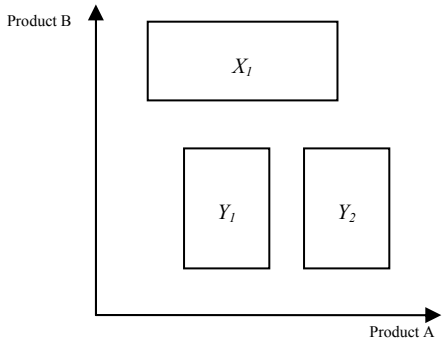


Figure 4.7: Initial Buffer Regions

The incoming arc region $X$ is transformed by applying the *f(x)* function for the node to create the transformed region, $Y'$. The transformed region $Y'$ is then intersected with the original outgoing regions $\mathcal{Y} = Y_1 \bigcup Y_2$ or $\mathcal{Y} = \{Y_1, Y_2\}$ to create the new outgoing regions $Y_{new}$. Note that the script font indicates a set of regions, of which each set is hyperrectangular.

$$Y' = f(X)$$

$$Y_{new,1} = Y' \bigcap Y_1$$

$$Y_{new,2} = Y' \bigcap Y_2$$

The new regions are intersected with the original regions to shrink the regions as the final solution is approached. The intersection is shown in the diagram below in which the original regions are outlined with solid lines and the transformed regions are outlined with dashed lines. A newly created region will be deleted if there is no intersection with the original regions. A new region could split into two regions if it were to overlap with two of the existing regions.

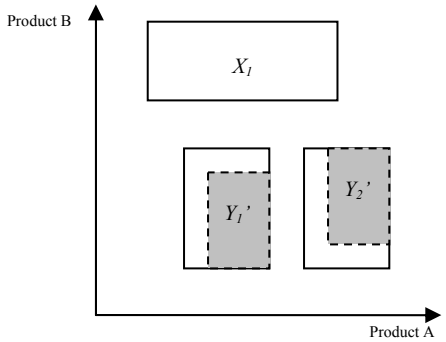


Figure 4.8: Intersection of Transformed Buffer Regions

The updated regions are shown in the following diagram.

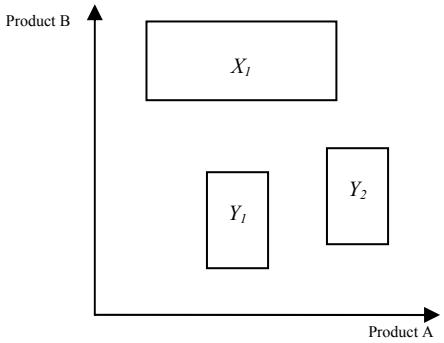


Figure 4.9: New Buffer Regions

The final step is to remove duplicate and subset regions from the newly created regions for each arc in an attempt to minimize the number of regions to be evaluated at each node. This operation is referred to as a merging of the regions because is not a true union in which any overlapping regions are joined into one region. The merging operation removes subset regions from the set of regions to create the final set of regions

$$\mathcal{Y}_{final} = Y_{new,1} \bigcup Y_{new,2}$$

or

$$\mathcal{Y}_{final} = \{Y_{new,1}, Y_{new,2}\}$$

The pseudo-inverse transformation of the outgoing regions follows the same procedure that has been discussed for the incoming regions. The outgoing regions $Y_1$ and $Y_2$ are inversely transformed by the node and then intersected with the original $X$ region to create a new $X'$ region. The merging of all the intersected regions is then calculated.

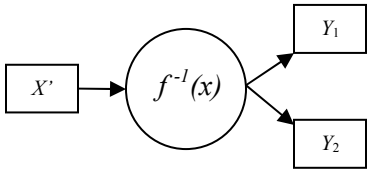


Figure 4.10: Transforming Outgoing Regions

$$X' = f^{-1}(Y)$$

$$X_{new,1} = X_1' \bigcap X$$

$$X_{new,2} = X_2' \bigcap X$$

$$\mathcal{X}_{final} = X_{new,1} \bigcup X_{new,2}$$

$$\mathcal{X}_{final} = \{X_{new,1}, X_{new,2}\}$$

## *4.5 Algorithm*

The stability algorithm uses a series of nested loops to cycle through the node network to determine stable product value regions for the system. For a given node the algorithm will first update the outgoing regions by transforming the incoming regions, and then intersecting and merging the new regions with the original outgoing regions. After all outgoing regions have been updated, the algorithm updates all the incoming regions by inversely transforming the outgoing regions and then intersecting and merging the new regions with the incoming regions. The node on each side of an arc is flagged by the algorithm only if a region has changed. After all regions are evaluated the algorithm will move to the next flagged node and repeat the evaluation of the regions. The algorithm evaluates the arc-node network by cycling through the flagged nodes of the network until there are no longer any flagged nodes, initially all nodes are flagged.

### 4.5.1 Stability Algorithm Outline

The stability algorithm requires the user to define the system parameters and network within the initialization function to accurately represent a given production system. The algorithm will then

initialize the product buffers for all arcs. The evaluation of the arc-node network begins after the initialization of the algorithm.

For a given node $i$, each product region $Y_j(a_{out})$ on each outgoing arc $a_{out}$ is examined first, where $j$ is the region sheet index number ($j = 1$, when only one region exists on the arc) and $a_{out} \in A_{out}(i)$. The algorithm uses a pair of nested loops to cycle through each outgoing region $Y_j(a_{out})$ for all outgoing arcs. Within the nested loops are another pair of nested loop that cycles through all incoming arcs $a_{in}$, and all incoming regions $X_j(a_{in})$ on each arc, where $a_{in} \in A_{in}(i)$.

$$\mathcal{X} = \bigcup_{j \in J} X_j(a_{in})$$

Each incoming region $X_j$ for a given incoming arc is transformed using the transformation function defined for the node to create the $Y'$ region.

$$Y' \leftarrow f(X, [t_{min}, t_{max}], i)$$

The $Y'$ region is stored in the set $\mathcal{Y}'$ and then newly transformed $Y'$ region is checked to verify that the region intersects with the original set of outgoing regions $\mathcal{Y}_{org}(a_{out})$. The algorithm then searches for a non-intersecting region, if found the $Y'$ region is flagged. This loop continues to cycle until all incoming arc regions have been transformed to create all possible $Y'$ regions for the given $a_{out}$.

Each region of the newly transformed set of regions, $\mathcal{Y}'(a_{out})$, is then intersected with the original set of outgoing arc regions $\mathcal{Y}_{org}$, and the regions are stored in the set $\mathcal{Y}_{\text{new}}$.

$$Y_{new} = Y' \bigcap Y_{org}$$

$$\mathcal{Y}_{new} = \{Y_{new}\}$$

Note that a non-intersected original region is maintained on the arc because a different incoming region that has not yet been transformed may intersect with the outgoing region. A non-intersecting region is deleted after all incoming regions have been transformed, intersected, and merged together. The new sets of outgoing regions are then cleaned up to remove duplicates,

subsets, or empty sets. The merging of the regions is then calculated and cleaned up for the outgoing arc region. The new set of outgoing arc regions, $\mathcal{Y}_{new}$ are compared with the original regions $\mathcal{Y}_{org}$ and if different, the node is flagged and will be reevaluated in the future. This process is repeated in a loop for the next region of the same outgoing arc if one exists, if not, the next outgoing arc is evaluated.

After exiting the outgoing arc loop, the non-intersecting regions are examined. The algorithm allows the user to define the *backward_stability* parameter as zero for forward stability checking only or set the parameter to one for determining stability in both directions. Solving for stability solely in the forward transformation direction, means that any point on a region will map forward to some other region and that point will then map forward to another region, etc. When the algorithm determines stability in both directions, both the forward transformations and backward inverse transformations must map to another region for each region. A non-intersecting incoming region is deleted if it has been flagged as not intersecting with any outgoing regions. If backward stability is being considered, the non-intersecting outgoing region then evaluated. An outgoing region is deleted if it has been flagged as not intersecting with any incoming regions

While still at the same node, the algorithm enters a set of nested loops to evaluate each incoming arc(s) and region(s) in the same fashion as the outgoing regions were evaluated. The algorithm cycles through the product regions $\mathcal{Y}$ for all outgoing arcs $a_{out} \in Arc_{out}(i)$, and the regions are inversely transformed, intersected with the incoming regions $X_i$, and merged in the same manner as the incoming arcs for the same node. The new set of incoming arc regions $X_i$, are compared with the original regions and if different, the node is flagged for future evaluation.

$$\mathcal{Y} = \bigcup_{j \in J} Y_j(a_{out})$$

Each outgoing region $Y_j$ for a given outgoing arc is inversely transformed using the inverse transformation function defined for the node to create the *X'* region.

$$X' \leftarrow f^{-1}(Y, [t_{min}, t_{max}], i)$$

The *X'* region is stored in the set $\mathcal{X}$' and then newly transformed *X'* region is checked to verify that the region intersects with the original set of incoming regions $\mathcal{X}_{org}(a_{in})$. The algorithm then

searches for a non-intersecting region, if found the *X'* region is flagged. This loop continues to cycle until all outgoing arc regions have been inversely transformed to create all possible *X'* regions for the given $a_{in}$.

Each region of the newly transformed set of regions, $\mathcal{X}'(a_{out})$, is then intersected with the original set of incoming arc regions $\mathcal{X}_{org}$, and the regions are stored in the set $\mathcal{X}_{\text{new}}$.

$$X_{new} = X' \bigcap X_{org}$$

$$\mathcal{X}_{new} = \{X_{new}\}$$

The new sets of incoming regions are then cleaned up to remove duplicates, subsets, or empty sets. The merging of the regions is then calculated and cleaned up for the incoming set of regions. The new set of incoming arc regions, $\mathcal{X}_{new}$ are compared with the original regions $\mathcal{X}_{org}$ and if different, the node is flagged and will be reevaluated in the future. This process is repeated in a loop for the next region of the same incoming arc if one exists, if not, the next incoming arc is evaluated.

The algorithm is ready to move to the next node that is flagged. This process continues at each flagged node until the regions no longer change when evaluated by the algorithm and there are no longer any flagged nodes.

### 4.5.2 Stability Algorithm Pseudo Code

The stability algorithm operates by calling functions to perform the required tasks. A brief description of the major variables and functions follows.

Variables:

- *flagged_node_list*: variable that stores a list of nodes to be evaluated by the algorithm. Initially all nodes are listed to be evaluated by the algorithm.
- $\mathcal{X}(arc)$ and $\mathcal{Y}(arc)$: variables for each arc that represent sets of hyperrectangular regions of the product buffer region. For a given arc, $\mathcal{X}(arc)$ and $\mathcal{Y}(arc)$ represent the same list of regions but is referred to as $\mathcal{X}$ or $\mathcal{Y}$ depending on whether the arc is an incoming arc or an outgoing arc for the node that is being evaluated.

- $X \in \mathcal{X}(arc)$ and $Y \in \mathcal{Y}(arc)$: the italic notation denotes an individual region within the sets of script variable, i.e. $X$ is a single region contained within $\mathcal{X}(arc)$.
- *no_intersection_flag*(*X,Y,arc,i*): binary variable in structured matrix used to flag a non-intersecting transformed (or inversely transformed) region. A value of one implies no n-intersecting region and value of zero implies an intersecting transformed region.
- *backward_stability*: this flag allows the user to consider solely forward stability (a region must map forward to another region but not required to map backward to another region, when *backward_stability* = 1) or to consider forward and backward stability (a region must map forward to another region and map backward to another region, when *backward_stability* = 1).
- *counter*: this variable records the number of cycles that have been completed by the algorithm to allow the user to define a stopping point for an unstable system.

Functions:

- initialization(): function to define various parameters of a given production system. The parameters defined by the function include the arc connections between the nodes, setup times, production rates, usage rates, buffer levels and thresholds, initial buffer limits, node types, and product-node associations.
- initialize_arc_limits(): function to define initial limits of regions based on node type and trigger or non-trigger product for the given node. For example the lower limit of each region on each arc in and out of an idle node is set to the lower threshold of the given product.
- set_node_time(*X,Y,i*): function that calculates the time spent at the current node *i* based on the current region sheet of the incoming arc *x* and outgoing arc region sheet *y*. The equations used for the time calculation were discussed in Sections 4.3.1 Forward Transformation Functions and 4.3.2 Pseudo-Inverse Transformation Functions.
- $f\left(X,[t_{\min},t_{\max}],i\right)$: the forward transformation function for node *i* for the incoming region *x* with the given time interval [$t_{min}$,$t_{max}$]. This function depends on the type of node, see Section 4.3.1 Forward Transformation Functions for a thorough discussion.
- $f^{-1}\left(Y,[t_{\min},t_{\max}],i\right)$: the pseudo-inverse transformation function for node *i* for the outgoing region *y* with the given time interval [$t_{min}$,$t_{max}$]. This function depends on the type of node, see Section 4.3.2 Pseudo-Inverse Transformation Functions for a thorough discussion.

- cleanup_merge_sheets(): operates on a given set ($X_{new}$ or $Y_{new}$) of regions input to the function. The function removes subsets, duplicate regions, empty sets, regions with minimum values greater than maximum values for a given product, and single point regions. The function also searches for two distinct regions in dimension $k$ that are equivalent in $k - 1$ dimensions, which can be represented as a single region that is the union of the two regions.
- update_change_flag(*i,change_flag,flagged_node_list*): this function adds the incoming or outgoing node to the list of nodes to be evaluated by the algorithm if current node has been flagged.
- update_node_number(*flagged_node_list*): this function updates the flagged node list by removing the current node from the list and all other nodes on the list are moved up the list. The first node on the list will be the next node to be evaluated by the algorithm.

The stability algorithm uses the following pseudo code to generate the stable regions:

```
1   define number_of_products
2   initialization(number_of_products)
3   initialize_arc_limits()
4   while (flagged_node_list ≠ Ø) and (counter < maximum number of cycles)
5           pick node i from flagged_node_list
6           for each a_out ∈ Arc_out(i)
7                   𝒴_new ← 0
8                   change_flag ← 0
9                   𝒴_org(a_out) = 𝒴(a_out)
10                  for each region Y ∈ 𝒴(a_out)
11                          for each a_in ∈ Arc_in(i)
12                                  for each region X ∈ 𝒳(a_in)
13                                          [t_min,t_max] ← set_node_time(X,Y,i)
14                                          Y' ← f(X, [t_min, t_max], i)
15                                          put Y' in list 𝒴'
16                                          if (Y' ∩ Y'' = Ø ) for each Y'' ∈ 𝒴_org
17                                                  no_intersection_flag(X,Y_org,arc_in,i) = 1
18                                          else
19                                                  no_intersection_flag(X,Y_org,arc_in,i) = 0
20                                          end
21                                  end
22                          end
23                  end
24                  for each region Y' ∈ 𝒴' and each Y_org ∈ 𝒴_org(a_out)
25                          Y_new = Y' ⋂ Y_org
26                          put Y_new in list 𝒴_new
```

```
27                  end
28                  cleanup_merge_sheets(𝒴_new)
29                  for each region Y_new ∈ 𝒴_new and each Y_org ∈ 𝒴_org(a_out)
30                        if Y_new ≠ Y_org(a_out) then
31                              change_flag = 1
32                        end
33                  end
34                  update_change_flag(i,change_flag,flagged_node_list)
35            end
36            for all a_in ∈ Arc_in(i) and all X ∈ 𝒳(a_in)
37                  if (no_intersection_flag(X,Y_org,arc_in,i) = 1),
38                        remove region X from 𝒳(a_in)
39                  end
40            end
41            if (backward_stability = 1)
42                  for all a_out ∈ Arc_out(i) and all Y_org ∈ 𝒴_org(a_out)
43                        if (no_intersection_flag(X,Y_org,arc_in,i) = 1),
44                              remove region Y_org from 𝒴_org(a_out)
45                        end
46                  end
47            end
48            for each a_in ∈ Arc_in(i)
49                  𝒳_new ← 0
50                  change_flag ← 0
51                  𝒳_org(a_out) = 𝒳(a_out)
52                  for each region X ∈ 𝒳(a_in)
53                        for each a_out ∈ Arc_out(i)
54                              for each region Y ∈ 𝒴(a_out)
55                                    [t_min,t_max] ←set_node_time(X,Y,i)
56                                    X'← f^-1 (Y, [t_min, t_max], i)
57                                    put X' in list 𝒳'
58                                    if (X' ∩X'' = Ø ) for each X'' ∈ 𝒳_org
59                                          no_intersection_flag(X_org,Y,arc_out,i) = 1
60                                    else
61                                          no_intersection_flag(X_org,Y,arc_out,i) = 0
62                                    end
63                              end
64                        end
65                  end
66                  for each region X' ∈ 𝒳' and each X_org ∈ 𝒳_org(a_in)
67                        X_new = X' ⋂ X_org
68                        put X_new in list 𝒳_new
69                  end
70                  cleanup_merge_sheets(𝒳_new)
71                  for each region X_new ∈ 𝒳_new and each X_org ∈ 𝒳_org(a_in)
72                        if X_new ≠ X_org(a_in) then
```

```
73                                  change_flag = 1
74                          end
75                  end
76                  update_change_flag(i,change_flag,flagged_node_list)
77          end
78          for all a_in ∈ Arc_in(i) and all X_org ∈ 𝒳_org(a_in)
79                  if (no_intersection_flag(X_org,Y,arc_in,i) = 1),
80                          remove region X_org from 𝒳_org(a_in)
81                  end
82          end
83          if (backward_stability = 1)
84                  for all a_out ∈ Arc_out(i) and all Y ∈ 𝒴(a_out)
85                          if (no_intersection_flag(X_org,Y,arc_in,i) = 1),
86                                  remove region Y from 𝒴(a_out)
87                          end
88                  end
89          end
90          update_node_number(flagged_node_list)
91          counter = counter +1
92  end
```

## *4.6 Implementation of Algorithm*

This algorithm was implemented using MATLAB Release 14, Version 7.0.4. The code of algorithm is in Appendix I.

## *4.7 Computational Complexity and Other Algorithm Issues*

The computational complexity of the algorithm has not been calculated formally but a quick review of the pseudo code reveals that the algorithm is not optimized. The algorithm was developed based on the overall function of the code instead of reducing computational time.

An example of the function of the code taking precedence over the computational efficiency is the set of nested loops used to remove non-intersecting flagged regions. These loop cycle through a set of matrices for all the arcs to search for flagged regions to be removed. This process requires multiple loops to search each matrix individually for the flagged regions and the results are evaluated and then the regions are removed. A more efficient method could be to store the flagged regions in a single matrix to remove one or more of the required nested loops.

### 4.7.1 Oscillating Regions

Overlapping regions can sometimes oscillate back and forth causing the algorithm to continue until the maximum counter value is reached. These oscillating regions do not change the area

covered by the set of regions, but merely change the representation of the region. Consider the dashed line in the following two plots.

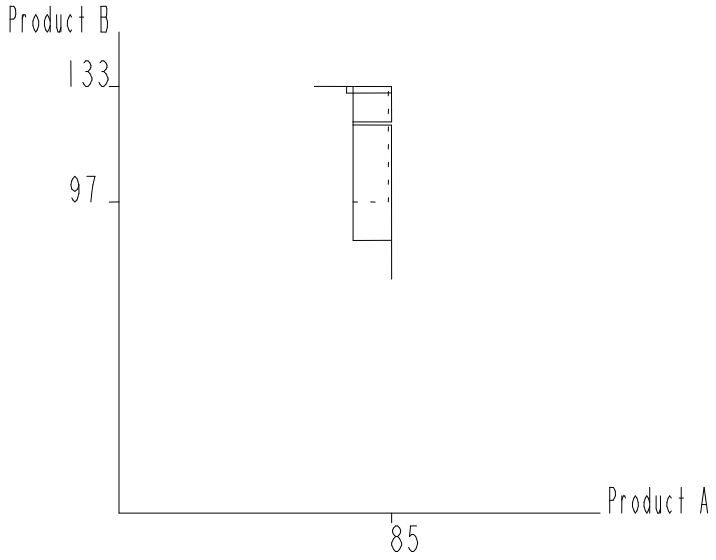


Figure 4.11: Oscillating Region – Configuration A

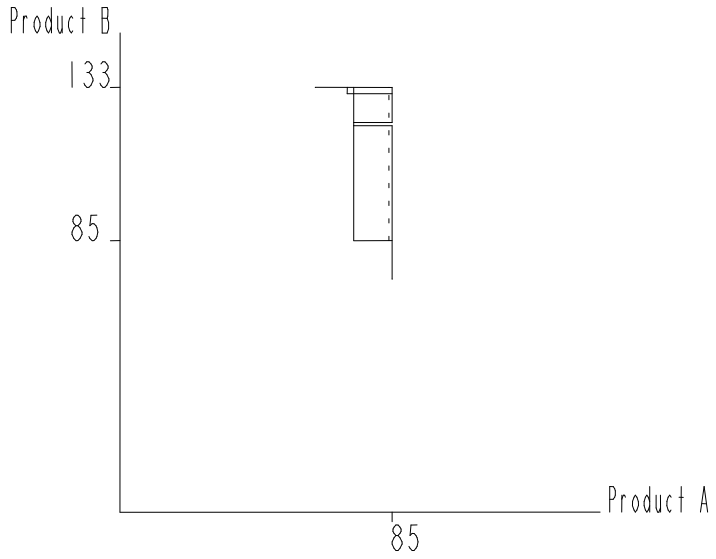


Figure 4.12: Oscillating Region – Configuration B

The oscillating region maintains a constant minimum value of 73 and a maximum value of 84 for product A. The maximum value of 133 for product B also does not change during the oscillation. The minimum value of product B for the oscillating region is either 85 or 97. Notice that this is the only region that is changing in the set of regions for this arc. The area that is lost in configuration B is still contained in another region within the set of regions.

# 5 Stability Algorithm Examples

This chapter is intended to provide insight into the stability algorithm by examining the output from the algorithm for various arc-node networks. A step-by-step example of a two-product system is the first example to be discussed followed by additional two-product systems. Three and four product systems are then discussed in this chapter.

## *5.1 Step-by-Step Example*

Consider a system for two products with arc-node network shown below and the following parameters: setup time = 5 time units, production rate = 10 products/time unit, usage rate = 1 product/time unit, lower threshold = 30 products, full buffer level = 100 products and idle time must exist.

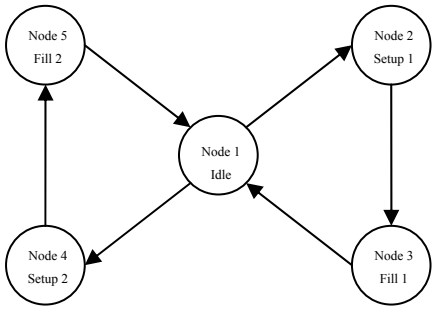


Figure 5.1: Network Map of Two-Product System – Idle Only

### 5.1.1 Initialization of Algorithm

A user can evaluate this system by first defining the number of products for the system. The algorithm then calls the *initialization* function which is responsible for defining setup times, production rates, usage rates, lower thresholds, full buffer levels, the arc-node network, node types, and node-product associations.

After initializing the system parameters and network, the algorithm calls the *initialize_arc_limits* function. This function defines the initial domain of each arc region based on the types of connected nodes and product-node associations. For example, the minimum product buffer levels are set to the lower threshold value, $l(k)$, for all arcs out of the idle node number one, arcs [1 2] and [1 4].

### 5.1.2 Flagged Node List Loop

The algorithm steps into the *flagged_node_list* loop which cycles through all the nodes in the network based on those that were initially flagged or are flagged by the algorithm during processing. Initially, all nodes are flagged to be evaluated. As the regions of the system begin to stabilize and no longer change, the nodes associated with the unchanged arc are removed from the flagged list. The algorithm will stop when there are no longer any nodes flagged to be evaluated. The flagged node loop also has a parameter *counter* for the maximum number of cycles that

allows the user to set an upper limit on the number of cycles for the algorithm to prevent an unstable system from running an infinite number of cycles.

The node number to be evaluated is picked from the *flagged_node_list* variable. The example network described above would start with node number one and the *flagged_node_list* variable would initially be [1 2 3 4 5]. The algorithm then will start to analyze the arcs connected to node one, which has two incoming arcs [3 1] and [5 1] and two outgoing arcs [1 2] and [1 4].

### 5.1.3 Outgoing Arc Analysis Loop

The algorithm steps into the outgoing arc loop to cycle through all the possible outgoing arcs contained in $Arc_{out}(1)$, which for this example $Arc_{out}(1) = \{[1\ 2], [1\ 4]\}$. Once inside the loop, the required variables are defined for each cycle through the loop. A nested loop is then entered to cycle through all regions $\mathcal{Y}(a_{out})$ associated with the current outgoing arc, $a_{out} = [1\ 2]$. Recall that the regions are stored as sheets of $k \times 2$ matrices on each arc, where $k$ is the number of products. Initially when the algorithm is started, each arc only has one product inventory region associated with it but more may be created as regions split from multiple incoming or outgoing arcs at a single node.

The algorithm then steps into two additional nested loops which cycle through all incoming arcs $a_{in} \in Arc_{in}(1)$ and all regions $\mathcal{X}(a_{in})$ on each incoming arc. For the current example network, the first outgoing arc [1 2] is evaluated first for the [3 1] incoming arc and then the algorithm loop advances to the second incoming arc, [5 1]. The $X \in \mathcal{X}(a_{in})$ loop has no effect initially in this example because there is only one region $X$ for all the incoming arcs.

The incoming arc group of nested loops allows the algorithm to transform all associated regions for each incoming arc to a new outgoing arc region. This allows the algorithm to calculate all possible combinations of trajectories of product sequencing for the newly transformed regions. Often many regions are duplicates, subsets, or supersets, but are still calculated by the algorithm for thoroughness.

Inside the incoming arc region loop (the $X \in \mathcal{X}(a_{in})$ loop) the *set_node_time* function is called to define the amount of time that is consumed at a given node. This function calculates the time based upon the trigger product number and all the minimum and maximum product region values.

The time spent at a setup node is the time to changeover from the previous trigger product to the current trigger product. A maximum and minimum time is computed for the refill node, where the maximum time is the time required to fill the trigger product from the lowest incoming region value to a full buffer level. The minimum refill time is the time required to fill the trigger product from the highest incoming region value to a full buffer level. See sections 3.1 Forward Transformation Functions and 3.2 Pseudo-Inverse Transformation Functions in the Stability Algorithm Chapter for a complete discussion of the time and transformation functions.

The current incoming and outgoing region values are used by the node transformation functions.

$$X = \begin{bmatrix} 100 & 100 \\ 30 & 100 \end{bmatrix}$$

$$Y = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

The algorithm then steps into a loop that cycles through each product in the system to determine a new outgoing region *Y'* using the node transformation functions for the given node type. After applying the node transformation functions to the region values for all products, the loop is exited.

Given that arc [1 2] is connected to the setup node for product one implies that product one is the trigger product. The transformation functions create the following *Y'* region for arc [1 2], where the top row is for product one and the bottom row is for product two:

$$Y' = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

The $\mathcal{Y}'$ set is updated with the newly created outgoing region *Y'*. The *arc_star* variable is a temporary variable that is used by the algorithm to store the *Y'* or *X'* values prior to calculating the intersection of the new regions. Each newly transformed region is tested to check if the transformed region *Y'* intersects with an original outgoing arc region $Y_{org} \in \mathcal{Y}_{\text{org}}$. If no intersection is found, the *X* region is flagged using the *no_intersection_flag* variable.

The incoming arc region loop is then indexed to the next region if possible or exited and the incoming arc loop is indexed. The incoming arc loop is indexed until all incoming arcs and regions have been used to find all possible new outgoing regions for the current outgoing arc and region.

For this example system, the incoming arc loop advances to the next incoming arc to evaluate arc $a_{in}$ = [5 1]. The *Y* matrix is the same as before but the *X* matrix is as follows:

$$X = \begin{bmatrix} 30 & 100 \\ 100 & 100 \end{bmatrix}$$

The transformation of the incoming arc [5 1] adds another sheet to the *arc_star* temporary variable, so it will now contain two regions:

$$\mathscr{Y}' = \{arc\_star(1,2,1), arc\_star(1,2,2)\}$$

$$arc\_star(1,2,1) = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

$$arc\_star(1,2,2) = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

The outgoing arc region loop is then indexed to the next outgoing region, if possible, and all incoming arcs and regions are again cycled through by the nested loops. After all possible outgoing regions have been evaluated for an outgoing arc; the outgoing arc region loop is exited. At this point all possible new product value regions have been stored in the *arc_star* variable for a single outgoing arc, which may or may not have multiple regions. In this example no other arcs or regions are available so the incoming arc region loop and the incoming arc region loop are both exited.

### 5.1.4 Intersect Function

The *intersect_arc_star* function is called to calculate the intersection of the original arc product value regions, which is the *arc_org* variable (equivalent to $\mathcal{Y}_{org}$), with the new product value regions, *arc_star* variable (equivalent to *Y'*). All new regions are intersected with each original region in an attempt to find all possible intersections. The newly intersected regions are stored in the *arc* variable (equivalent to $Y_{new}$) and the *arc_star* variable is erased.

In this example network, the product value regions of $a_{out}$ = [1 2] were originally

$$Y_{org} = \text{arc}(1,2) = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

The original region of *arc*(1,2) intersected with the two regions of *arc_star*(1,2) creates the following set of regions:

$$\mathcal{Y}_{new} = \{arc(1,2,1), arc(1,2,2)\}$$

$$arc(1,2,1) = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

$$arc(1,2,2) = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

The algorithm then calls the *cleanup_sheets* function for the current outgoing arc to remove duplicate, empty, or other improper regions, such as a minimum region value being greater than the maximum region value. For this example the function removes one of the duplicate regions on the *arc*(1,2).

$$\mathcal{Y}_{new} = \{arc(1,2)\}$$

$$arc(1,2) = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

### 5.1.5 Merge Arc Function

The cleaned *arc* variable is then operated upon by the *merge_arc* function, although in this example there is only one region to operate on, so the single region will be unchanged by the function. The *merge_arc* function determines the merging, if possible, of the regions for the current arc by checking for supersets, subsets, and equivalent sets. The function also calculates the union of two unique regions in $k$ dimensions, if the two regions share boundaries in $k - 1$ dimensions and overlap in $k$ directions, where $k$ is number of products. The function updates the *arc* variable ($\mathcal{Y}_{new}$) which is then cleaned again using the *cleanup_sheets* function. Arc [1 2] is unchanged after calling the *merge_arc* and *cleanup_sheets* functions and has the following product value region:

$$arc(1,2) = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

The algorithm then calls the *set_change_flag* function to compare the updated set of arc regions ($\mathcal{Y}_{new}$) with the original set of arc regions prior to applying the transformation equations ($\mathcal{Y}_{org}$). If the regions are equivalent the *change_flag* remains equal to zero. If one or more regions are different from the original regions, the current outgoing arc is flagged as having changed. The regions for the current arc [1 2] have not changed from the original values so node number one and node number two are not flagged for future evaluation. The *update_change_flag* function is then called. This function adds nodes from each end of the flagged arc to the list to be evaluated if the nodes are not already listed on the flagged node list to be revisited in the future by the algorithm.

At this point the outgoing arc loop is indexed to the next outgoing arc if one exists and the previously described process is repeated or the loop is exited if there are no more outgoing arcs. For the example node network, the outgoing arc loop is indexed to the outgoing arc [1 4] at node one. This arc will be evaluated for both incoming arcs [3 1] and [5 1]. Note that the *arc_star* variable is a temporary variable that is used by the algorithm to store the *Y'* or *X'* values prior to calculating the intersection of the new regions. Upon completion of the outgoing arc region loop, *arc_star(1,4)* variable will contain the following matrices:

$$\mathcal{Y}' = \{arc\_star(1,4,1), arc\_star(1,4,2)\}$$

$$arc\_star(1,4,1) = \begin{bmatrix} 30 & 100 \\ 30 & 30 \end{bmatrix}$$

$$arc\_star(1,4,2) = \begin{bmatrix} 30 & 100 \\ 30 & 30 \end{bmatrix}$$

After calling the *intersect_arc_star*, *cleanup_sheets*, and *merge_arc* functions, the $\mathcal{Y}_{new}$([1 4]) regions will have been intersected, cleaned up, and merged with the *arc_star* regions to create the following product region values:

$$\mathcal{Y}_{new} = \{arc(1,4)\}$$

$$arc(1,4) = \begin{bmatrix} 30 & 100 \\ 30 & 30 \end{bmatrix}$$

Upon exiting the outgoing arc loop, the algorithm will have updated all the regions for all the outgoing arcs connected to the current node. The $\mathcal{Y}_{new}$ sets of regions no longer have the *new* subscript and are simply $\mathcal{Y}$ sets of regions. If the regions changed from the original values, the nodes from each end of the outgoing arc will be listed on the *flagged_node_list* to be revisited in the future by the algorithm.

After exiting the outgoing arc loop, the algorithm evaluates the *no_intersection_flag* variable for forward stability by examining all incoming regions contained in $\mathcal{X}(Arc_{in})$. If an incoming region *X* is flagged as non-intersecting with an outgoing region *Y* for all transformations of the *X* region, the incoming *X* region is removed because it does not map forward to any of the outgoing regions contained in $\mathcal{Y}(Arc_{out})$. In this example system, no incoming regions are flagged or removed because both incoming regions map to one of the outgoing regions.

If the user is considering backward stability, the algorithm evaluates the *no_intersection_flag* variable for all outgoing regions contained in $\mathcal{Y}(Arc_{out})$. If an outgoing region *Y* is flagged as non-intersecting with an incoming region *X* for all transformations of the set of regions $\mathcal{X}(Arc_{in})$, the outgoing *Y* region is removed because it does not map backward to any of the incoming regions contained in $\mathcal{X}(Arc_{in})$. In this example system, no outgoing regions are flagged or removed because each of the outgoing regions maps backward to one of the incoming regions.

At this point, the product values regions for the outgoing arcs are as follows:

$$\mathcal{Y}\left(arc(1,2)\right)=\begin{bmatrix}30 & 30\\ 30 & 100\end{bmatrix}$$

$$\mathcal{Y}\left(arc(1,4)\right)=\begin{bmatrix}30 & 100\\ 30 & 30\end{bmatrix}$$

And the incoming arcs have the following values:

$$\mathcal{Y}\left(arc(3,1)\right)=\begin{bmatrix}100 & 100\\ 30 & 100\end{bmatrix}$$

$$\mathcal{Y}\left(arc(5,1)\right)=\begin{bmatrix}30 & 100\\ 100 & 100\end{bmatrix}$$

### 5.1.6 Incoming Arc Analysis

The algorithm now begins the same process of nested loops to update the regions for the incoming arcs $a_{in} \in Arc_{in}$ connected to the current node one. The incoming arc loop steps through all the incoming arcs connected to the current node that is being evaluated. Incoming arc [3 1] will be evaluated first in the example node network, followed by incoming arc [5 1]. The required parameters and variables associated with the current incoming arc are defined when the algorithm steps into the incoming arc loop. The algorithm then steps into the nested incoming arc region loop to cycle through all the regions associated with the current incoming arc, $X \in \mathcal{X}(a_{in})$. The next nested loop is the outgoing arc loop which is responsible for cycling through all the outgoing arcs, $a_{out} \in Arc_{out}$, connected to the current node.

The *set_node_time* function is called when the algorithm enters the outgoing arc region loop to define the amount of time that is consumed for a given node. The outgoing region to be used by the node inverse-transformation function for arc [1 2] is as follows:

$$Y = \begin{bmatrix} 30 & 30 \\ 30 & 100 \end{bmatrix}$$

The algorithm then steps into a loop that will cycle through each product in the system to calculate the inverse node transformation function for the given node type to create a new incoming region, *X'*. After applying the node transformation functions to the region values for all products in the given region, the loop is exited. The *arc_star* variable ($\mathcal{X}'$) is updated with the newly created outgoing region values. After updating the *arc_star* variable,

Each newly inversely transformed region is tested to check if the transformed region *X'* intersects with an original incoming arc region $X_{org} \in \mathcal{X}_{\text{org}}$. If no intersection is found, the *X* region is flagged using the *no_intersection_flag* variable.

The outgoing arc region loop is indexed to the next outgoing arc region if possible or the loop is exited and the outgoing arc loop is indexed. The outgoing arc region loop is indexed until all outgoing arcs and regions have been used to find all possible new incoming regions $\mathcal{X}'$ for the current incoming arc and region. The incoming arc region loop is then indexed to the next incoming region, if possible, and all outgoing arcs and regions are again cycled through by the nested loops. After all possible incoming regions have been evaluated for an incoming arc, the incoming arc loop is exited.

The example network will have the following *arc_star* product value regions after the algorithm has cycled through the incoming arc region loop and subsequent nested loops to inversely transform the outgoing arcs:

$$\mathcal{X}' = \{arc\_star(3,1,1), arc\_star(3,1,2)\}$$

$$arc\_star(3,1,1) = \begin{bmatrix} 100 & 100 \\ 30 & 100 \end{bmatrix}$$

$$arc\_star(3,1,2) = \begin{bmatrix} 100 & 100 \\ 30 & 100 \end{bmatrix}$$

At this point all possible new product value regions have been stored in the *arc_star* variable for a single incoming arc, which may or may not have multiple regions. The *intersect_arc_star* function is called to calculate the intersection of the original arc product value regions, $\mathcal{X}_{org}$, with the new regions, $\mathcal{X}'$, as was done previously in the outgoing arc loop. All new regions are intersected with each current region in an attempt to find all possible intersections, which are stored as $\mathcal{X}_{new}$, in the *arc* variable and the *arc_star* variable is erased. The algorithm then calls the *cleanup_sheets* function for the current incoming arc to remove duplicate, empty, or other improper region sets.

The cleaned up *arc* variable ($\mathcal{X}_{new}$) is then operated upon by the *merge_arc* function. This function performs the same merging of regions for the current incoming arc as was previously performed for the regions in the outgoing arc loop. The function updates the *arc* variable which is then cleaned up using the *cleanup_sheets* function. The merging and cleaning up of *arc* will produce the following product value region:

$$\mathcal{X}_{new} = \{arc(3,1)\}$$

$$arc(3,1) = \begin{bmatrix} 30 & 100 \\ 30 & 100 \end{bmatrix}$$

The algorithm then calls the *set_change_flag* function to compare the update arc regions, $\mathcal{X}_{new}$, with the arc regions prior to applying the transformation equations, $\mathcal{X}_{org}$. If the regions are equivalent the upstream and downstream nodes are not flagged but if the regions are different from the original regions, the node is flagged to be revisited if it is not already included in the flagged node list.

At this point the incoming arc loop is exited or indexed to the next incoming arc if one exists and the previously described process for the incoming arc is repeated. The current example network will be indexed to the second incoming arc [5 1], which will be transformed to the following region after the completion of the incoming arc loop.

$$\mathcal{X}_{new} = \{arc(5,1)\}$$

$$arc(5,1)=\begin{bmatrix}30 & 100\\ 30 & 100\end{bmatrix}$$

Upon exiting the incoming arc loop the algorithm will have updated all the regions for all the incoming arcs connected to the current node.

The algorithm has now updated the product value regions for both the outgoing arcs and incoming arcs for the current node number one. The node will not be flagged by the *set_change_flag* function because both the incoming and outgoing arc regions are unchanged.

The algorithm calls the *update_change_flag* function to update the list of nodes to be evaluated by the algorithm, which is stored in the *flagged_node_list* variable. After calling the *update_change_flag* function the *flagged_node_list* variable is [2 3 4 5].

After updating the *flagged_node_list* variable, the *update_node_number* function is called to change the node number to the next node in the *flagged_node_list* variable. After calling the *update_node_number* function the *flagged_node_list* variable is unchanged in this example. The algorithm proceeds to the next node and loops through the *flagged_node_list* loop, if the *flagged_node_list* is not empty; otherwise the algorithm stops the analysis.

The example node network will continue to node number two because it is the next node in the *flagged_node_list* variable. The same procedure of cycling through and transforming the outgoing and incoming arc regions continues as previously described. The *flagged_node_list* variable is now [3 4 5 2]. The output from the algorithm for node number two will update the product value regions as follows:

$$\mathcal{X}\left(arc(1,2)\right)=\begin{bmatrix}30 & 30\\ 30 & 100\end{bmatrix},\mathcal{Y}\left(arc(2,3)\right)=\begin{bmatrix}25 & 25\\ 25 & 95\end{bmatrix}$$

The algorithm next evaluates node number three in the example network and updates the product value regions of the incoming and outgoing arcs to the following:

$$\mathcal{X}\left(arc(2,3)\right)=\begin{bmatrix}25 & 25\\ 38 & 95\end{bmatrix},\mathcal{Y}\left(arc(3,1)\right)=\begin{bmatrix}100 & 100\\ 30 & 87\end{bmatrix}$$

The *flagged_node_list* variable is now [4 5 2 3 1]. Node number four is the next node to be evaluated by the algorithm. The updated product value regions for the node are the following:

$$\mathcal{X}\left(arc(1,4)\right)=\begin{bmatrix}30 & 100\\ 30 & 30\end{bmatrix},\mathcal{Y}\left(arc(4,5)\right)=\begin{bmatrix}25 & 95\\ 25 & 25\end{bmatrix}$$

After evaluation of node number four, the *flagged_node_list* variable is now [5 2 3 1 4] which requires node number five to be evaluated by the algorithm. The updated product value regions for node five are the following:

$$\mathcal{X}\left(arc(4,5)\right)=\begin{bmatrix}38 & 95\\ 25 & 25\end{bmatrix},\mathcal{Y}\left(arc(5,1)\right)=\begin{bmatrix}30 & 87\\ 100 & 100\end{bmatrix}$$

After evaluation of node number five, the *flagged_node_list* variable is now [2 3 1 4 5]. The algorithm will continue to loop through the node network until the *flagged_node_list* variable is empty and the regions no longer change when transformed by the algorithm. This example takes approximately three complete cycles through the network to find the stable product value regions. The final output for all the arcs are as follows:

$$arc(1,2)=\begin{bmatrix}30 & 30\\ 43 & 100\end{bmatrix},arc(2,3)=\begin{bmatrix}25 & 25\\ 38 & 95\end{bmatrix}$$

$$arc(3,1)=\begin{bmatrix}100 & 100\\ 30 & 87\end{bmatrix},arc(1,4)=\begin{bmatrix}43 & 100\\ 30 & 30\end{bmatrix}$$

$$arc(4,5)=\begin{bmatrix}38 & 95\\ 25 & 25\end{bmatrix},arc(5,1)=\begin{bmatrix}30 & 87\\ 100 & 100\end{bmatrix}$$

### *5.2 General Statements about Implementation of Algorithm*

The convergence of the algorithm is not guaranteed due to the chaotic nature of a switched arrival system, as demonstrated by Chase et al. [5]. The user can set a maximum number of iterations

for the algorithm to limit the maximum number of times that the node network will be cycled through to prevent it from running indefinitely.

The regions for an arc are always equivalent or subsets of the original regions when the algorithm completes the evaluation of a node. The regions will never increase in size, but instead a region may divide, shrink, or remain unchanged. In a very general sense, the purpose of the algorithm is to settle the regions on each side of a node until the regions are equivalent when transformed by the node. Any arc in the node network will contain at least one region set, possibly an empty set. Typically the number of regions for an outgoing arc will increase as the number of arcs entering the node increase.

## *5.3 Output from the Algorithm*

The output from the algorithm is a set of one or more regions for each defined arc in the system; it is possible that a set may be empty. A settled and stable system will always have a trajectory that can propagate from a point on one region which is transformed to a point on another region and any transformed point will always be contained within a defined region.

### 5.3.1 Two Product Network

Consider the previously discussed example system for two products with the node/arc network shown below and the following parameters: setup time = 5 time units, production rate = 10 products/time unit, usage rate = 1 product/time unit, lower threshold = 30 products, full buffer level = 100 products and idle time must exist.

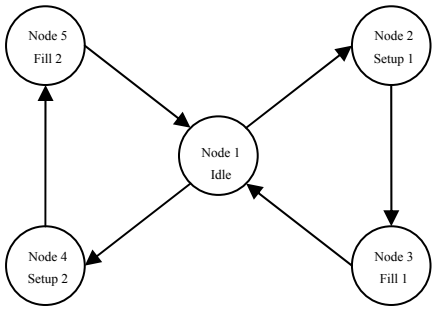


Figure 5.2: Network Map of Two-Product System –Idle Only

Consider the region for *arc(2,3)* below, which is for the region of products that are leaving the setup for product one at node number two and going to fill product one at node number three. Product one is represented by the top row and has a minimum of 25 and maximum of 25 products

for the region. The region in the second row is for product two and has a minimum value of 38 and maximum value of 95.

$$arc(2,3) = \begin{bmatrix} 25 & 25 \\ 38 & 95 \end{bmatrix}$$

The region of [25 25] and [38 95] is representative of the values that the products must fall within when leaving node number two and entering node number three to be stable. If the products are within the region, then by the results in Section 5 of the Node Network Analysis Chapter, the system will remain stable for all time, given that variability, breakdowns, maintenance, etc. never occur in this system. In this example, the region is a one dimensional line, but a region may be a point, a line, two-dimensional surface, or multidimensional volume dependent upon the parameters of the system and number of products. Typically the product value regions are of $k$ – 1 dimensions because one product will always have equivalent minimum and maximum values, i.e. one-dimensional lines represent two product regions, two-dimensional rectangular planes for three product systems, etc.

As previously discussed, this two product example network takes approximately three complete cycles through the network to find the stable product value regions which are as follows:

$$arc(1,2) = \begin{bmatrix} 30 & 30 \\ 43 & 100 \end{bmatrix}, arc(2,3) = \begin{bmatrix} 25 & 25 \\ 38 & 95 \end{bmatrix}$$

$$arc(3,1) = \begin{bmatrix} 100 & 100 \\ 30 & 87 \end{bmatrix}, arc(1,4) = \begin{bmatrix} 43 & 100 \\ 30 & 30 \end{bmatrix}$$

$$arc(4,5) = \begin{bmatrix} 38 & 95 \\ 25 & 25 \end{bmatrix}, arc(5,1) = \begin{bmatrix} 30 & 87 \\ 100 & 100 \end{bmatrix}$$

#### 5.3.1.1 Stable Two-Product Network

Consider the plot below where the solid lines represent the regions and the dashed lines represent the connections between the extents of the regions for the flow direction of the products from arc region to arc region. The lines with arrowheads represent a stable orbit and direction that the system operates for a given starting point. From this plot it is very apparent that the system will

never crash by experiencing an empty buffer, because the system parameters cause the regions to be located far from the origin of the plot [0, 0], where both product buffers are completely empty.

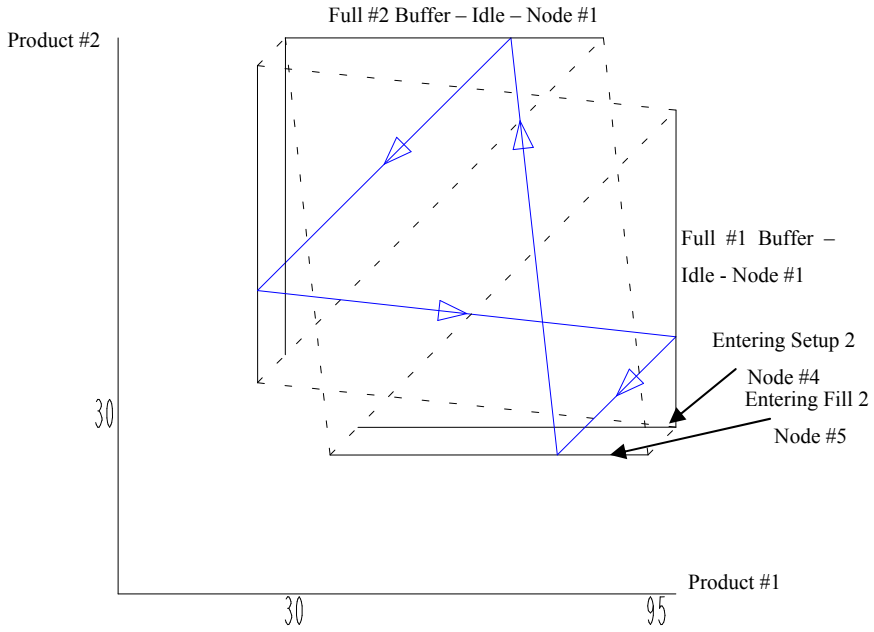


Figure 5.3: Plot of Two-Product Network

The dashed lines in the plot connect the extents of a region to the next region and have a slope of the cumulative usage rate of the products when travelling from a full buffer to a setup node. The slope of the line between the filling node and idle node is the refill rate of the product being replenished divided by the cumulative usage rates of all products. A better illustration of the dashed line is to imagine that pairs of parallel dashed lines represent a node and the lines show how the products are transformed by the idle, setup, or a refill node. When the dashed lines connect the extents of each region to another, the system can never leave the stable regions. The only possible way for the system to leave the stable orbit is for some type of variability to disrupt the system, which is not considered in the current work.

The algorithm is a conservative estimate of the stable product regions for a node network system. Regions that are defined by the stability algorithm will guarantee stability for the defined set of values for the products. But the stable regions are not guaranteed to be inclusive of all stable values for the products. Note that the algorithm is searching for long term stability and does not consider start-up conditions or non-steady state conditions of the system. Due to ignoring transient conditions, it is quite possible for the system to initially operate well outside of a stable

region but eventually crossover into a stable region and therefore a stable orbit, which it will never leave. If at any point in time the orbit lands on any of the defined regions for the system, the orbit of the production system will always be stable into the future. Any point on any stable region will always map onto another stable region of the system and the orbit will therefore always be stable in the future. See Node Network Analysis Chapter, Section 5 for a thorough discussion and proof of this statement.

Consider the heavy long-dashed lines in the plots below which represent three possible trajectories (A, B, and C) of the system. The trajectories all start outside of the stable regions, as defined by the stability algorithm. These trajectories highlight that the system would easily reach a stable region after one or two cycles of refilling the products. This example also highlights that for this given system, the stable regions are very conservative, meaning that the regions are not inclusive of all possible stable product value combinations.

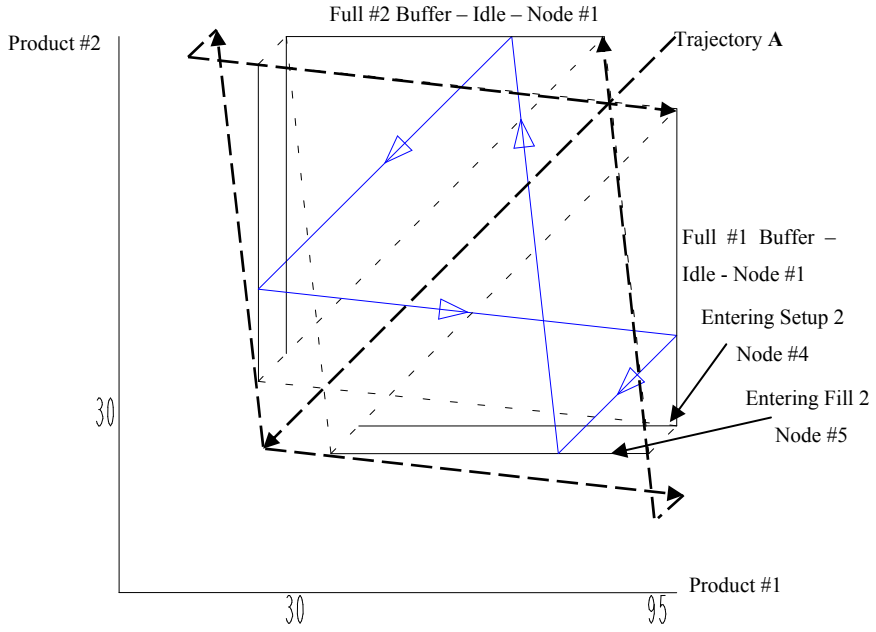


Figure 5.4: Plot of Non-Included Trajectory A

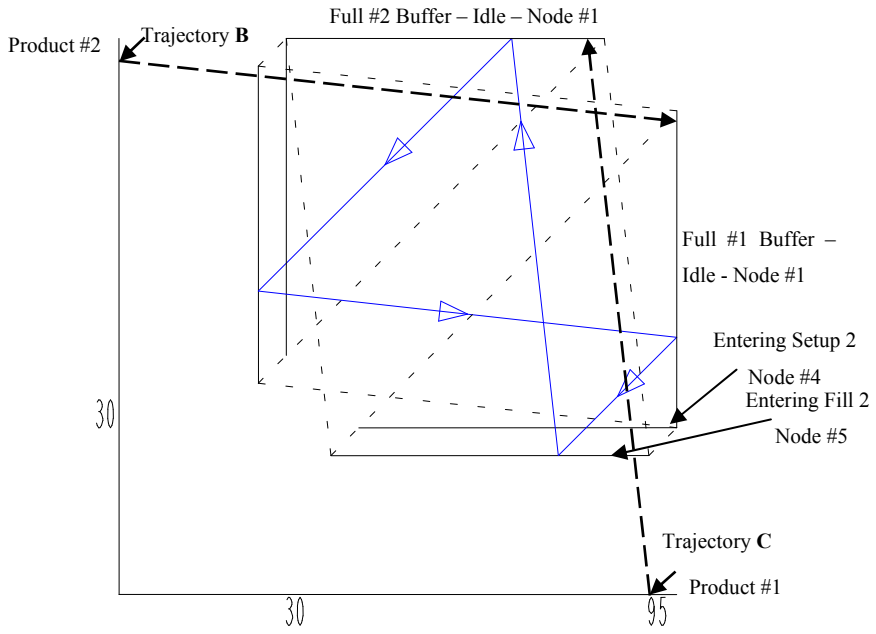


Figure 5.5: Plot of Non-Included Trajectories B and C

5.3.1.2 Unstable Two-Product Network

Consider the same node network and parameters previously discussed with the only exception being that the usage rate of product one is doubled to consuming two products per time unit, *rho*(1) = 2 and *rho*(2) = 1. This slight change in system parameters causes the system to be unstable with no possible stable regions existing and the algorithm outputs all empty sets. The following regions are a snapshot of the system during the process of attempting to find stable regions. A plot of the intermediate regions highlights the instability issue for the regions.

$$arc(1,2,1)=\begin{bmatrix}30 & 30\\ 45 & 50\end{bmatrix}, arc(1,2,2)=\begin{bmatrix}30 & 30\\ 78 & 83\end{bmatrix}, arc(2,3,1)=\begin{bmatrix}20 & 20\\ 73 & 78\end{bmatrix}$$

$$arc(2,3,2)=\begin{bmatrix}20 & 20\\ 40 & 45\end{bmatrix}, arc(3,1,1)=\begin{bmatrix}100 & 100\\ 30 & 35\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 63 & 68\end{bmatrix}$$

$$arc(1,4)=\begin{bmatrix}90 & 100\\ 30 & 30\end{bmatrix}, arc(4,5)=\begin{bmatrix}80 & 90\\ 25 & 25\end{bmatrix}, arc(5,1)=\begin{bmatrix}63 & 73\\ 100 & 100\end{bmatrix}$$

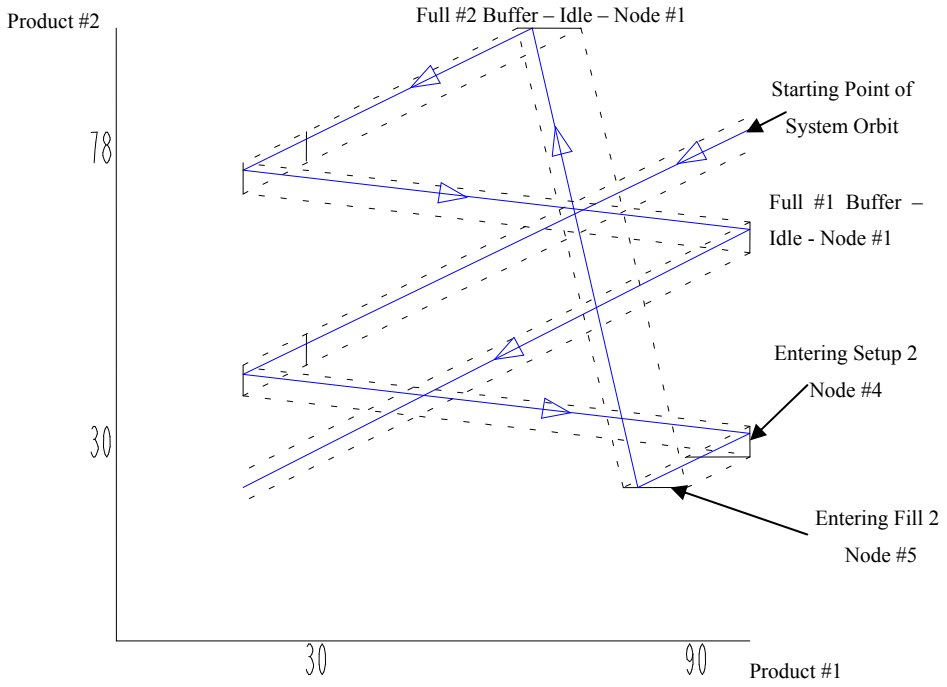


Figure 5.6: Two-Product System – Product 1 Doubled Usage Rate

The solid black lines are the intermediate regions and the short-dashed lines are the connections between the extents of the regions based on the refill rates or usage rates. The plotted path of a potential system orbit highlights that the paths eventually reach a location after the second refill of product number one. This highlights how the trajectory does not transform from region to region and is therefore unstable.

#### 5.3.1.3 Increasing the Product One Threshold

Raising the lower threshold for product number one to 40 units is a minor change in the system and is easily tested as a possible solution for the unstable system. The output from the algorithm is again empty sets for the production system with a lower threshold limit of 40 units for product number one and 30 units for product number two. An intermediate set of regions prior to the algorithm settling the network are below.

$$arc(1,2)=\begin{bmatrix}40 & 40\\ 44 & 56\end{bmatrix}, arc(1,2)=\begin{bmatrix}40 & 40\\ 83 & 96\end{bmatrix}, arc(2,3,1)=\begin{bmatrix}30 & 30\\ 78 & 91\end{bmatrix}$$

$$arc(2,3,2)=\begin{bmatrix}30 & 30\\ 39 & 51\end{bmatrix}, arc(3,1,1)=\begin{bmatrix}100 & 100\\ 30 & 43\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 70 & 82\end{bmatrix}$$

$$arc(1,4)=\begin{bmatrix}75 & 100\\ 30 & 30\end{bmatrix}, arc(4,5)=\begin{bmatrix}65 & 90\\ 25 & 25\end{bmatrix}, arc(5,1)=\begin{bmatrix}48 & 73\\ 100 & 100\end{bmatrix}$$

A plot of these regions is below. The orbit is unstable and again falls outside of the regions after product number one has been refilled. From the plot of intermediate regions, it is apparent the system cannot be stable for this network configuration given the current parameters.

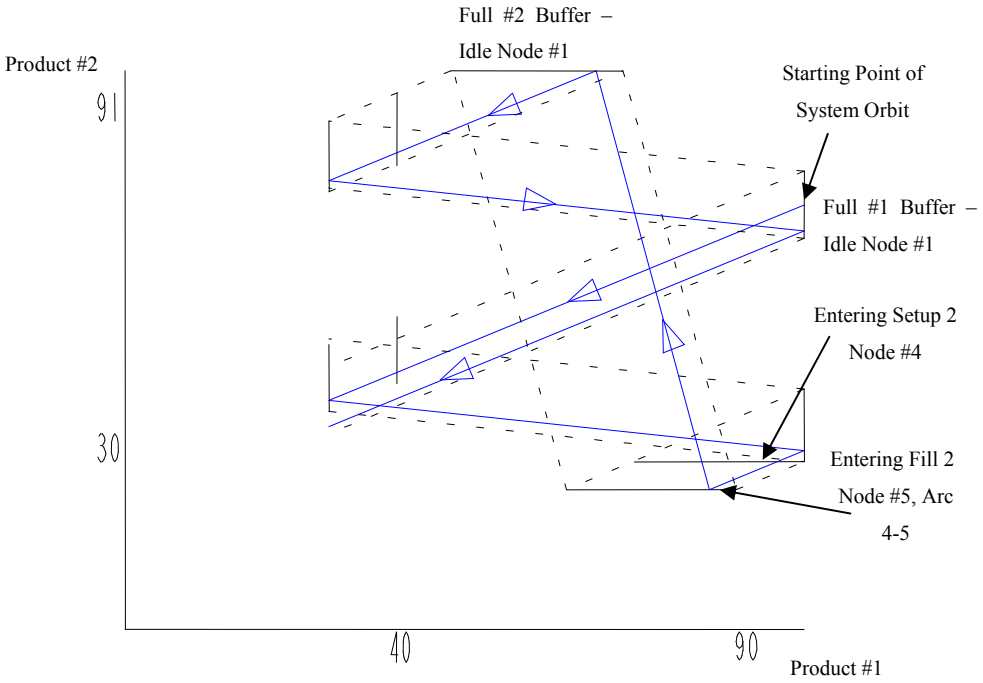


Figure 5.7: Two-Product System – Product 1 Threshold of 40 Products

#### 5.3.1.4 Two-Product System With and Without Idle

Consider the following node network diagramed below, which has the same parameters as the initial two-product example system except that an arc now connects each refill node to the setup node of the other product. The node network now allows the system to skip the idle node if a product crosses the lower threshold prior to completely replenishing the buffer of the trigger product.

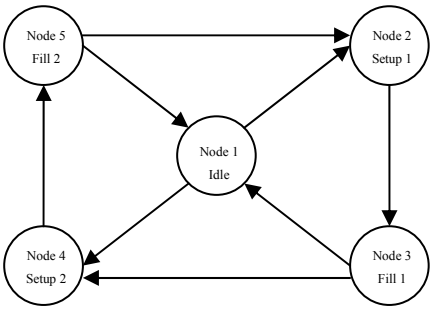


Figure 5.8: Network Map of Two-Product System – With and Without Idle

Examining this system with a lower limit of 30 units for both products and a usage rate of two for product number one and a usage rate of one for product number two will again output empty set regions. Consider the following intermediate regions which when plotted highlight the cause of the instability.

$$arc(1,2)=\begin{bmatrix}30 & 30\\ 30 & 50\end{bmatrix}, arc(1,2)=\begin{bmatrix}30 & 30\\ 78 & 83\end{bmatrix}, arc(5,2)=\begin{bmatrix}10 & 30\\ 100 & 100\end{bmatrix}$$

$$arc(2,3,1)=\begin{bmatrix}0 & 20\\ 95 & 95\end{bmatrix}, arc(2,3,2)=\begin{bmatrix}20 & 20\\ 73 & 78\end{bmatrix}, arc(2,3,3)=\begin{bmatrix}20 & 20\\ 25 & 45\end{bmatrix}$$

$$arc(3,1,1)=\begin{bmatrix}100 & 100\\ 83 & 85\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 63 & 68\end{bmatrix}, arc(3,1,3)=\begin{bmatrix}100 & 100\\ 30 & 35\end{bmatrix}$$

$$arc(1,4)=\begin{bmatrix}90 & 100\\ 30 & 30\end{bmatrix}, arc(3,4)=\begin{bmatrix}100 & 100\\ 15 & 30\end{bmatrix}, arc(4,5,1)=\begin{bmatrix}90 & 90\\ 10 & 25\end{bmatrix}$$

$$arc(4,5,2)=\begin{bmatrix}80 & 90\\ 25 & 25\end{bmatrix}, arc(5,1)=\begin{bmatrix}63 & 73\\ 100 & 100\end{bmatrix}$$

A plot of these intermediate regions is shown below. The system crashes after product one is refilled because the idle node transforms the incoming region to a nonexistent outgoing region in the lower left hand corner of the plot. Without any intersection between the transformed and existing regions, the resulting region is an empty set. The empty set is propagated through the system with subsequent transformations at each node, resulting in an empty set for all arcs. This

indicates that there is no stable trajectory for this arc-node network, given the current set of parameters.

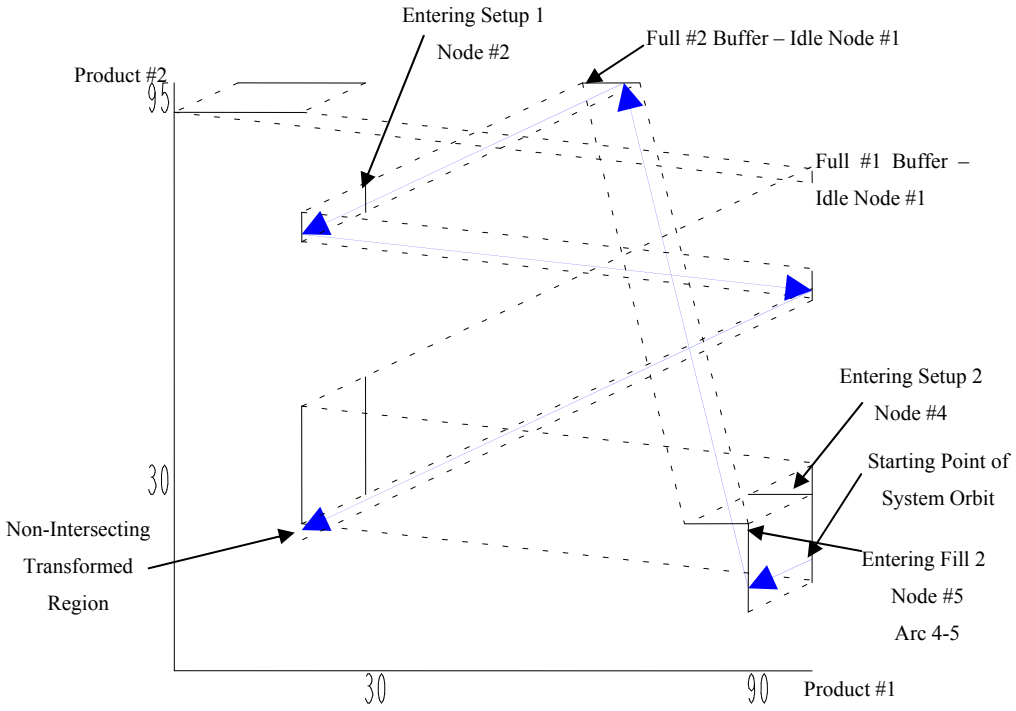


Figure 5.9: Two-Product System – With and Without Idle – 30 Unit Threshold

Consider increasing the lower threshold for product one to 50 units to increase the buffer depth when the product triggers replenishment. The system parameters are now a lower limit value of 50 units and usage rate of two for product number one and a lower limit of 30 units and usage rate of one for product number two. This system will provide the output regions listed below.

$$arc(1,2,1)=\begin{bmatrix}50 & 50\\ 51 & 52\end{bmatrix}, arc(1,2,2)=\begin{bmatrix}50 & 50\\ 59 & 61\end{bmatrix}, arc(1,2,3)=\begin{bmatrix}50 & 50\\ 97 & 98\end{bmatrix}, arc(5,2)=\begin{bmatrix}36 & 40\\ 100 & 100\end{bmatrix}$$

$$arc(2,3,1)=\begin{bmatrix}26 & 30\\ 95 & 95\end{bmatrix}, arc(2,3,2)=\begin{bmatrix}40 & 40\\ 83 & 85\end{bmatrix}, arc(2,3,3)=\begin{bmatrix}40 & 40\\ 92 & 93\end{bmatrix}, arc(2,3,4)=\begin{bmatrix}40 & 40\\ 46 & 47\end{bmatrix}$$

$$arc(2,3,5)=\begin{bmatrix}40 & 40\\ 54 & 56\end{bmatrix}, arc(3,1,1)=\begin{bmatrix}100 & 100\\ 84 & 86\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 76 & 77\end{bmatrix}, arc(3,1,3)=\begin{bmatrix}100 & 100\\ 47 & 49\end{bmatrix}$$

$$arc(3,1,3)=\begin{bmatrix}100 & 100\\ 38 & 40\end{bmatrix}, arc(1,4,1)=\begin{bmatrix}81 & 83\\ 30 & 30\end{bmatrix}, arc(1,4,2)=\begin{bmatrix}63 & 67\\ 30 & 30\end{bmatrix}, arc(3,4)=\begin{bmatrix}100 & 100\\ 18 & 30\end{bmatrix}$$

$$arc(4,5,1)=\begin{bmatrix}90 & 90\\ 13 & 25\end{bmatrix}, arc(4,5,1)=\begin{bmatrix}71 & 73\\ 25 & 25\end{bmatrix}, arc(4,5,2)=\begin{bmatrix}53 & 57\\ 25 & 25\end{bmatrix}, arc(5,1,1)=\begin{bmatrix}71 & 73\\ 100 & 100\end{bmatrix}$$

$$arc(5,1,2)=\begin{bmatrix}54 & 57\\ 100 & 100\end{bmatrix}$$

A plot of the regions is below. All regions of the system now map forward to another region. This system will remain stable for all orbits that originate from any location on a defined output region. Notice that from any starting point on a stable region, the system is able to reach the final stable orbit within a few cycles of replenishing both products.

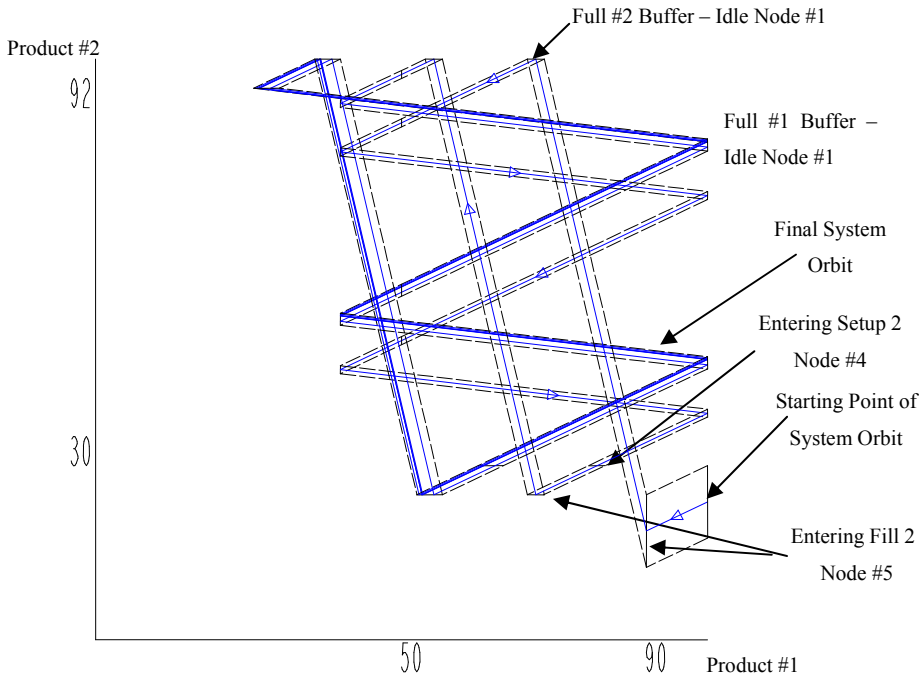


Figure 5.10: Two-Product System – With and Without Idle – Product One 50 Unit Threshold

This example system with the give set of parameters provides an interesting insight into the stability algorithm. Notice in the plot above in Figure 5.10, the region in the lower right corner has no incoming region connection. This region is on the arc [3 4], which in the arc that connects the fill node for product number one and the setup node for product number two. This means that

none of the regions on arc [2 3] entering the refill node map forward to the region on arc [3 4]. These results are when only forward stability is being considered by the algorithm. The algorithm allows the user to solve for stability solely in the forward transformation direction or in both directions with the forward transformations and backward inverse transformations. The user may define the *backward_stability* parameter as zero for forward checking only or set the parameter to one for stability in both directions. Note that when the stability algorithm checks for stability in both directions, the algorithm will output a different set of regions for this example problem, which are listed below.

$$arc(1,2)=\begin{bmatrix}50 & 50\\ 61 & 61\end{bmatrix}, arc(5,2)=\begin{bmatrix}36 & 37\\ 100 & 100\end{bmatrix}$$

$$arc(2,3,1)=\begin{bmatrix}26 & 27\\ 95 & 95\end{bmatrix}, arc(2,3,2)=\begin{bmatrix}40 & 40\\ 56 & 56\end{bmatrix}$$

$$arc(3,1,1)=\begin{bmatrix}100 & 100\\ 48 & 48\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 86 & 86\end{bmatrix}$$

$$arc(1,4)=\begin{bmatrix}63 & 64\\ 30 & 30\end{bmatrix}, arc(3,4)=[\varnothing]$$

$$arc(4,5)=\begin{bmatrix}53 & 54\\ 25 & 25\end{bmatrix}, arc(5,1)=[\varnothing]$$

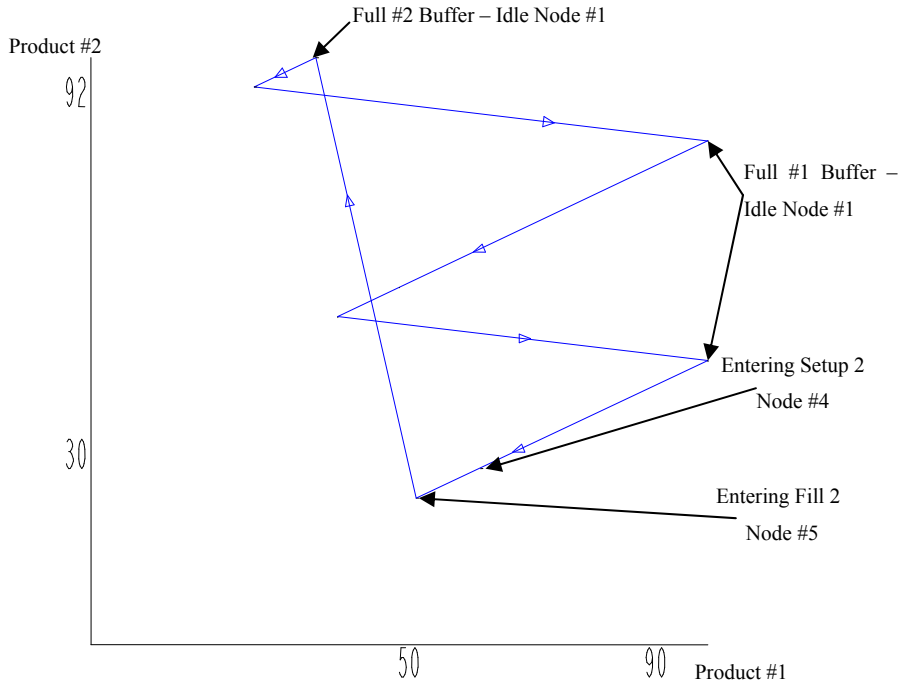


Figure 5.11: Product One 50 Unit Threshold – Forward and Backward Stability

### 5.3.2 Three Product Network

The computational complexity will increase with each additional product added to the arc-node network because each additional product will add at least two nodes (setup and fill) and three arcs to the existing network. Consider a system with three products with the arc-node network show below and the following parameters: setup time = 5 time units, production rate = 10 products/time unit, usage rate = 1 product/time unit, lower threshold = 30 products, full buffer level = 100 products and idle time must exist.

#### 5.3.2.1 Three Product Network – Idle Only

In this example network, all product sequences must pass through the idle node prior to entering setup as shown in the following diagram.

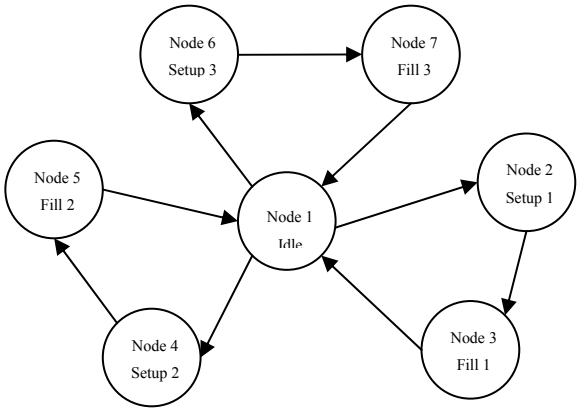


Figure 5.12: Network Map of Three-Product System –Idle Only

Determining the stable product value regions for this node network is very straight forward and could be conducted by hand because the product parameters are all equal and the system is balanced. The results for the stable regions from the algorithm are as below.

$$arc(1,2,1)=\begin{bmatrix}30 & 30\\ 57 & 100\\ 43 & 87\end{bmatrix}, arc(1,2,2)=\begin{bmatrix}30 & 30\\ 43 & 87\\ 57 & 100\end{bmatrix}, arc(2,3,1)=\begin{bmatrix}25 & 25\\ 52 & 95\\ 38 & 82\end{bmatrix}, arc(2,3,2)=\begin{bmatrix}25 & 25\\ 38 & 82\\ 52 & 95\end{bmatrix}$$

$$arc(3,1,1)=\begin{bmatrix}100 & 100\\ 43 & 87\\ 30 & 73\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 30 & 73\\ 43 & 87\end{bmatrix}, arc(1,4,1)=\begin{bmatrix}57 & 100\\ 30 & 30\\ 43 & 87\end{bmatrix}, arc(1,4,2)=\begin{bmatrix}43 & 87\\ 30 & 30\\ 57 & 100\end{bmatrix}$$

$$arc(4,5,1)=\begin{bmatrix}52 & 95\\ 25 & 25\\ 38 & 82\end{bmatrix}, arc(4,5,2)=\begin{bmatrix}38 & 82\\ 25 & 25\\ 52 & 95\end{bmatrix}, arc(5,1,1)=\begin{bmatrix}43 & 87\\ 100 & 100\\ 30 & 73\end{bmatrix}, arc(5,1,2)=\begin{bmatrix}30 & 73\\ 100 & 100\\ 43 & 87\end{bmatrix}$$

$$arc(1,6,1)=\begin{bmatrix}57 & 100\\ 43 & 87\\ 30 & 30\end{bmatrix}, arc(1,6,2)=\begin{bmatrix}43 & 87\\ 57 & 100\\ 30 & 30\end{bmatrix}, arc(6,7,1)=\begin{bmatrix}52 & 95\\ 38 & 82\\ 25 & 25\end{bmatrix}, arc(6,7,2)=\begin{bmatrix}38 & 82\\ 52 & 95\\ 25 & 25\end{bmatrix}$$

$$arc(7,1,1)=\begin{bmatrix}43 & 87\\ 30 & 73\\ 100 & 100\end{bmatrix}, arc(7,1,2)=\begin{bmatrix}30 & 73\\ 43 & 87\\ 100 & 100\end{bmatrix}$$

Examination of the output regions highlights that each region is a 2-dimensional plane because the minimum and maximum value of one product is always equal for each region. The output from the stability produces regions of $k - 1$ dimensions.

Consider the regions defined for *arc(1,2,1)* and *arc(1,2,2)* in which product number one is equal to 30 units for both regions. The extents of the regions are show below plotted as a rectangle on a plane through all points where product number one is equal to 30 units. The heavy arrow-headed lines show the direction of the total product usage rate for the system in two dimensions (the usage rates for all products is one product per time unit).

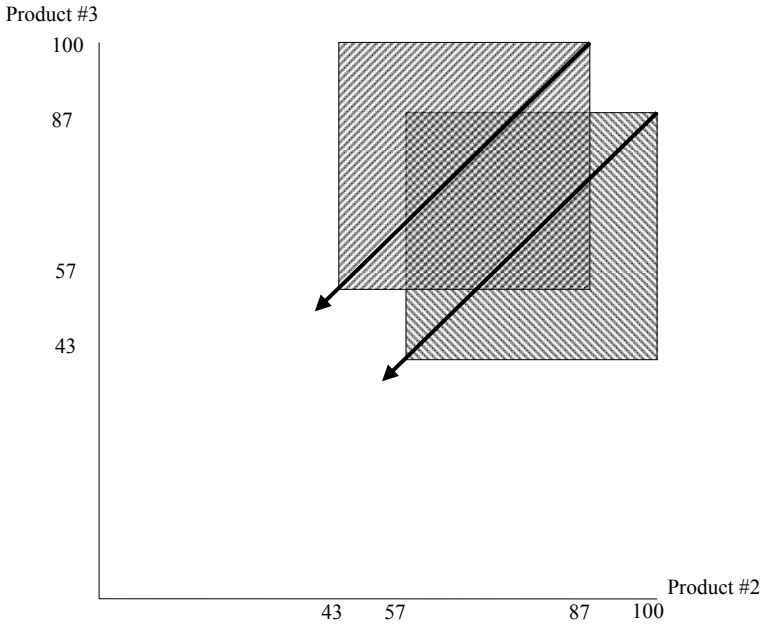


Figure 5.13: Two-Dimensional Plot of Arc [1 2] Regions

The area of the regions between the diagonal lines creates instability problems when a sequence is traced through the production system that originates within this area. A discontinuity exists when the center region points transition from a full buffer region to a region that is leaving the idle node because the central region of a full buffer region is transformed by the idle node into an empty region. This can be seen in the following diagram which is a three dimensional plot with the regions represented as rectangles.

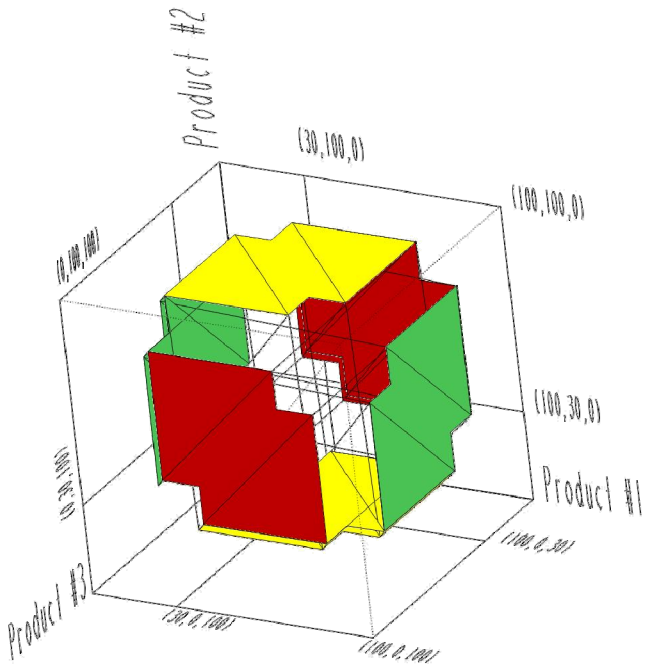


Figure 5.14: Three-Dimensional Plot of Rectangular Arc Regions

The system parameters cause the total usage rate for the system to be parallel to a line with a slope of one in the X, Y, and Z directions. Consider the top surface in the region below which represents when product number two has a full buffer of 100 units. As the system is idle and products are consumed the trajectory must hit either the region below it on the left (green) where product number one is equal to the lower threshold of 30 units or the region on the right (red) where product number three is equal to the lower threshold of 30 units for the system to remain stable. The area between the two triangular shaped areas does not map to a region.

The following plot of the regions has been corrected to remove the unstable portion of the regions defined by the system. For this plot it is easy to see that a product sequence trajectory can start at any point on any of the defined regions and the trajectory will always transform to another stable region for all possible node transformations that may occur in the node network of the system.

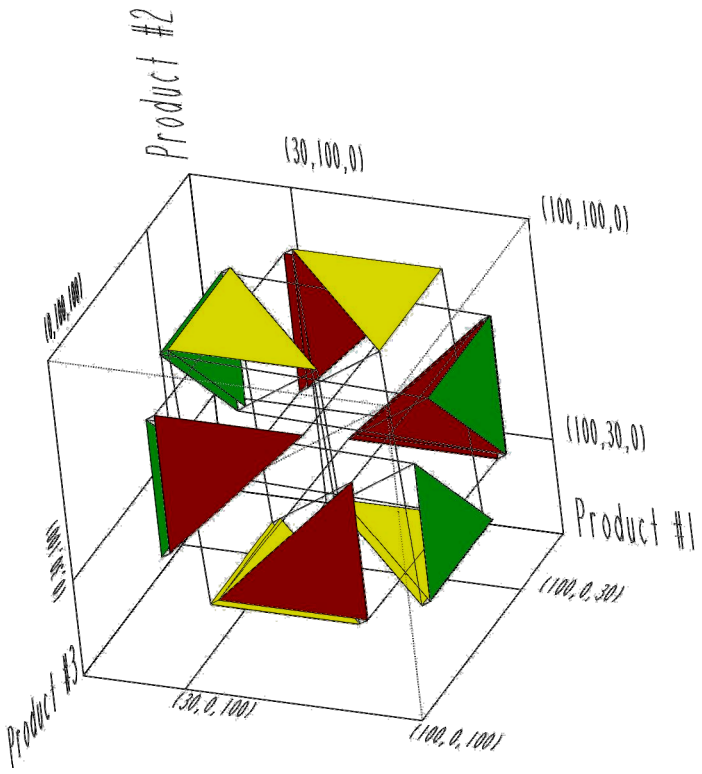


Figure 5.15: Three-Dimensional Plot of Triangular Arc Regions

This example highlights the need for some interpretation or special handling of the regions for systems with more than two products. In a system in which the parameters are defined in such a way that the usage rates, buffer sizes, production rates, and lower thresholds of every product are equal, the stable regions will be triangular in shape bounded by the cumulative usage rate in two-dimensions. The proposed solution to correct this discontinuity issue for more complex production system is define the network in such a way that idle time is not required to follow every refill node, which will remove the discontinuities between setup regions.

#### 5.3.2.2 Three Product Network – With and Without Idle

Consider a system with three products with the node/arc network show below and the same parameters as the previous example; setup time = 5 time units, production rate = 10 products/time unit, usage rate = 1 product/time unit, lower threshold = 30 products, full buffer level = 100 products. In this example network, all product sequences may pass through the idle node prior to being replenished or a product can enter setup directly after replenishment of the previous product.

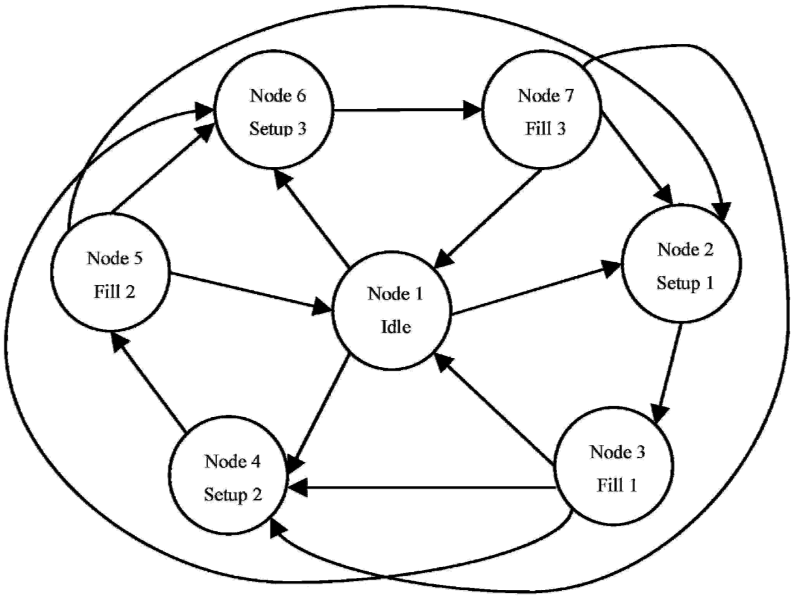


Figure 5.16: Network Map of Three-Product System – With and Without Idle

The algorithm cycles through the network approximately twelve times to determine the stable regions for the system. The results from the algorithm for this arc-node network still contain the same regions as the previous system, but these regions a now segmented into smaller regions. The output also contains the additional regions for the arcs that skip idle. The regions for a given arc is symmetric with two other arcs, meaning that all regions on arc [1 2] are the same values as arcs [1 4] and [1 6], but on a different plane. For example, consider the following region on arc [1 2]:

$$arc(1,2,1) = \begin{bmatrix} 30 & 30 \\ 59 & 89 \\ 46 & 83 \end{bmatrix}$$

This region on arc [1 2] is symmetric with the following regions on arcs [1 4] and [1 6]:

$$arc(1,4,1) = \begin{bmatrix} 59 & 89 \\ 30 & 30 \\ 46 & 83 \end{bmatrix}, arc(1,6,1) = \begin{bmatrix} 59 & 89 \\ 46 & 83 \\ 30 & 30 \end{bmatrix}$$

The output regions for the system are below.

Arc [1 2]:

$$arc(1,2,1)=\begin{bmatrix}30 & 30\\ 59 & 89\\ 46 & 83\end{bmatrix}, arc(1,2,2)=\begin{bmatrix}30 & 30\\ 59 & 98\\ 59 & 83\end{bmatrix}, arc(1,2,3)=\begin{bmatrix}30 & 30\\ 60 & 99\\ 46 & 84\end{bmatrix}, arc(1,2,4)=\begin{bmatrix}30 & 30\\ 46 & 84\\ 46 & 54\end{bmatrix},$$

$$arc(1,2,5)=\begin{bmatrix}30 & 30\\ 46 & 83\\ 46 & 54\end{bmatrix}, arc(1,2,6)=\begin{bmatrix}30 & 30\\ 46 & 83\\ 59 & 89\end{bmatrix}, arc(1,2,7)=\begin{bmatrix}30 & 30\\ 59 & 83\\ 59 & 97\end{bmatrix}, arc(1,2,8)=\begin{bmatrix}30 & 30\\ 46 & 84\\ 60 & 99\end{bmatrix},$$

$$arc(1,2,9)=\begin{bmatrix}30 & 30\\ 46 & 54\\ 46 & 84\end{bmatrix}, arc(1,2,10)=\begin{bmatrix}30 & 30\\ 46 & 54\\ 46 & 83\end{bmatrix}, arc(1,2,11)=\begin{bmatrix}30 & 30\\ 30 & 35\\ 59 & 97\end{bmatrix}, arc(1,2,12)=\begin{bmatrix}30 & 30\\ 59 & 97\\ 30 & 35\end{bmatrix}$$

Arc [2 3]:

$$arc(2,3,1)=\begin{bmatrix}12 & 17\\ 95 & 95\\ 41 & 79\end{bmatrix}, arc(2,3,2)=\begin{bmatrix}12 & 17\\ 41 & 79\\ 95 & 95\end{bmatrix}, arc(2,3,3)=\begin{bmatrix}25 & 25\\ 54 & 84\\ 41 & 78\end{bmatrix}, arc(2,3,4)=\begin{bmatrix}25 & 25\\ 54 & 92\\ 54 & 78\end{bmatrix},$$

$$arc(2,3,5)=\begin{bmatrix}25 & 25\\ 55 & 94\\ 41 & 79\end{bmatrix}, arc(2,3,6)=\begin{bmatrix}25 & 25\\ 41 & 79\\ 41 & 49\end{bmatrix}, arc(2,3,7)=\begin{bmatrix}25 & 25\\ 41 & 78\\ 41 & 49\end{bmatrix}, arc(2,3,8)=\begin{bmatrix}25 & 25\\ 41 & 78\\ 54 & 84\end{bmatrix},$$

$$arc(2,3,9)=\begin{bmatrix}25 & 25\\ 54 & 78\\ 54 & 92\end{bmatrix}, arc(2,3,10)=\begin{bmatrix}25 & 25\\ 41 & 79\\ 55 & 94\end{bmatrix}, arc(2,3,11)=\begin{bmatrix}25 & 25\\ 41 & 49\\ 41 & 79\end{bmatrix}, arc(2,3,12)=\begin{bmatrix}25 & 25\\ 41 & 49\\ 41 & 78\end{bmatrix},$$

$$arc(2,3,13)=\begin{bmatrix}25 & 25\\ 25 & 30\\ 54 & 92\end{bmatrix}, arc(2,3,14)=\begin{bmatrix}25 & 25\\ 54 & 92\\ 25 & 30\end{bmatrix}$$

Arc [3 1]:

$$arc(3,1,1)=\begin{bmatrix}100 & 100\\ 46 & 76\\ 33 & 69\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 33 & 69\\ 46 & 76\end{bmatrix}, arc(3,1,3)=\begin{bmatrix}100 & 100\\ 46 & 84\\ 46 & 69\end{bmatrix},$$

$$arc(3,1,4)=\begin{bmatrix}100 & 100\\ 46 & 69\\ 46 & 84\end{bmatrix}, arc(3,1,5)=\begin{bmatrix}100 & 100\\ 33 & 71\\ 33 & 40\end{bmatrix}, arc(3,1,6)=\begin{bmatrix}100 & 100\\ 33 & 40\\ 33 & 71\end{bmatrix},$$

$$arc(3,1,7)=\begin{bmatrix}100 & 100\\ 85 & 86\\ 31 & 70\end{bmatrix}, arc(3,1,8)=\begin{bmatrix}100 & 100\\ 31 & 70\\ 85 & 86\end{bmatrix}, arc(3,1,9)=\begin{bmatrix}100 & 100\\ 33 & 70\\ 33 & 41\end{bmatrix},$$

$$arc(3,1,10)=\begin{bmatrix}100 & 100\\ 33 & 41\\ 33 & 70\end{bmatrix}, arc(3,1,11)=\begin{bmatrix}100 & 100\\ 47 & 85\\ 33 & 71\end{bmatrix}, arc(3,1,12)=\begin{bmatrix}100 & 100\\ 33 & 71\\ 47 & 85\end{bmatrix}$$

Arcs [3 4] and [3 6]:

$$arc(3,4)=\begin{bmatrix}100 & 100\\ 17 & 22\\ 46 & 84\end{bmatrix}, arc(3,6)=\begin{bmatrix}100 & 100\\ 46 & 84\\ 17 & 22\end{bmatrix}$$

Arc [1 4]:

$$arc(1,4,1)=\begin{bmatrix}59 & 89\\ 30 & 30\\ 46 & 83\end{bmatrix}, arc(1,4,2)=\begin{bmatrix}59 & 97\\ 30 & 30\\ 59 & 83\end{bmatrix}, arc(1,4,3)=\begin{bmatrix}60 & 99\\ 30 & 30\\ 46 & 84\end{bmatrix}, arc(1,4,4)=\begin{bmatrix}46 & 84\\ 30 & 30\\ 46 & 54\end{bmatrix},$$

$$arc(1,4,5)=\begin{bmatrix}46 & 83\\ 30 & 30\\ 46 & 54\end{bmatrix}, arc(1,4,6)=\begin{bmatrix}43 & 83\\ 30 & 30\\ 59 & 89\end{bmatrix}, arc(1,4,7)=\begin{bmatrix}59 & 83\\ 30 & 30\\ 59 & 97\end{bmatrix}, arc(1,4,8)=\begin{bmatrix}46 & 84\\ 30 & 30\\ 60 & 99\end{bmatrix},$$

$$arc(1,4,9)=\begin{bmatrix}46 & 54\\ 30 & 30\\ 46 & 84\end{bmatrix}, arc(1,4,10)=\begin{bmatrix}46 & 54\\ 30 & 30\\ 46 & 83\end{bmatrix}, arc(1,4,11)=\begin{bmatrix}30 & 35\\ 30 & 30\\ 59 & 97\end{bmatrix}, arc(1,4,12)=\begin{bmatrix}59 & 97\\ 30 & 30\\ 30 & 35\end{bmatrix},$$

Arc [4 5]:

$$arc(4,5,1)=\begin{bmatrix}95 & 95\\ 12 & 17\\ 41 & 79\end{bmatrix}, arc(4,5,2)=\begin{bmatrix}41 & 79\\ 12 & 17\\ 95 & 95\end{bmatrix}, arc(4,5,3)=\begin{bmatrix}54 & 84\\ 25 & 25\\ 41 & 78\end{bmatrix}, arc(4,5,4)=\begin{bmatrix}54 & 92\\ 25 & 25\\ 54 & 78\end{bmatrix},$$

$$arc(4,5,5)=\begin{bmatrix}55 & 94\\ 25 & 25\\ 41 & 79\end{bmatrix}, arc(4,5,6)=\begin{bmatrix}41 & 79\\ 25 & 25\\ 41 & 49\end{bmatrix}, arc(4,5,7)=\begin{bmatrix}41 & 78\\ 25 & 25\\ 41 & 49\end{bmatrix}, arc(4,5,8)=\begin{bmatrix}41 & 78\\ 25 & 25\\ 54 & 84\end{bmatrix},$$

$$arc(4,5,9)=\begin{bmatrix}54 & 78\\ 25 & 25\\ 54 & 92\end{bmatrix}, arc(4,5,10)=\begin{bmatrix}41 & 79\\ 25 & 25\\ 55 & 94\end{bmatrix}, arc(4,5,11)=\begin{bmatrix}41 & 49\\ 25 & 25\\ 41 & 79\end{bmatrix}, arc(4,5,12)=\begin{bmatrix}41 & 49\\ 25 & 25\\ 41 & 78\end{bmatrix},$$

$$arc(4,5,13)=\begin{bmatrix}25 & 30\\ 25 & 25\\ 54 & 92\end{bmatrix}, arc(4,5,12)=\begin{bmatrix}54 & 92\\ 25 & 25\\ 25 & 30\end{bmatrix}$$

Arc [5 1]:

$$arc(5,1,1)=\begin{bmatrix}46 & 76\\ 100 & 100\\ 33 & 69\end{bmatrix}, arc(5,1,2)=\begin{bmatrix}33 & 69\\ 100 & 100\\ 46 & 76\end{bmatrix}, arc(5,1,3)=\begin{bmatrix}46 & 84\\ 100 & 100\\ 46 & 69\end{bmatrix},$$

$$arc(5,1,4)=\begin{bmatrix}46 & 69\\ 100 & 100\\ 46 & 84\end{bmatrix}, arc(5,1,5)=\begin{bmatrix}33 & 71\\ 100 & 100\\ 33 & 40\end{bmatrix}, arc(5,1,6)=\begin{bmatrix}33 & 40\\ 100 & 100\\ 33 & 71\end{bmatrix},$$

$$arc(5,1,7)=\begin{bmatrix}85 & 86\\ 100 & 100\\ 31 & 70\end{bmatrix}, arc(5,1,8)=\begin{bmatrix}31 & 70\\ 100 & 100\\ 85 & 86\end{bmatrix}, arc(5,1,9)=\begin{bmatrix}33 & 70\\ 100 & 100\\ 33 & 41\end{bmatrix},$$

$$arc(5,1,10)=\begin{bmatrix}33 & 41\\ 100 & 100\\ 33 & 70\end{bmatrix}, arc(5,1,11)=\begin{bmatrix}47 & 85\\ 100 & 100\\ 33 & 71\end{bmatrix}, arc(5,1,12)=\begin{bmatrix}33 & 71\\ 100 & 100\\ 47 & 85\end{bmatrix}$$

Arcs [5 2] and [5 6]:

$$arc(5,2)=\begin{bmatrix}17 & 22\\ 100 & 100\\ 46 & 84\end{bmatrix}, arc(5,6)=\begin{bmatrix}46 & 84\\ 100 & 100\\ 17 & 22\end{bmatrix}$$

Arc [1 6]:

$$arc(1,6,1)=\begin{bmatrix}59 & 89\\ 46 & 83\\ 30 & 30\end{bmatrix}, arc(1,6,2)=\begin{bmatrix}59 & 97\\ 59 & 83\\ 30 & 30\end{bmatrix}, arc(1,6,3)=\begin{bmatrix}60 & 99\\ 46 & 84\\ 30 & 30\end{bmatrix}, arc(1,6,4)=\begin{bmatrix}46 & 84\\ 46 & 54\\ 30 & 30\end{bmatrix},$$

$$arc(1,6,5)=\begin{bmatrix}46 & 83\\ 46 & 54\\ 30 & 30\end{bmatrix}, arc(1,6,6)=\begin{bmatrix}46 & 83\\ 59 & 89\\ 30 & 30\end{bmatrix}, arc(1,6,7)=\begin{bmatrix}59 & 83\\ 59 & 97\\ 30 & 30\end{bmatrix}, arc(1,6,8)=\begin{bmatrix}46 & 84\\ 60 & 99\\ 30 & 30\end{bmatrix},$$

$$arc(1,6,9)=\begin{bmatrix}46 & 54\\ 46 & 84\\ 30 & 30\end{bmatrix}, arc(1,6,10)=\begin{bmatrix}46 & 54\\ 46 & 83\\ 30 & 30\end{bmatrix}, arc(1,6,11)=\begin{bmatrix}30 & 35\\ 59 & 97\\ 30 & 30\end{bmatrix}, arc(1,6,12)=\begin{bmatrix}59 & 97\\ 30 & 35\\ 30 & 30\end{bmatrix}$$

Arc [6 7]:

$$arc(6,7,1)=\begin{bmatrix}95 & 95\\ 41 & 79\\ 12 & 17\end{bmatrix}, arc(6,7,2)=\begin{bmatrix}41 & 79\\ 95 & 95\\ 12 & 17\end{bmatrix}, arc(6,7,3)=\begin{bmatrix}54 & 84\\ 41 & 78\\ 25 & 25\end{bmatrix}, arc(6,7,4)=\begin{bmatrix}54 & 92\\ 54 & 78\\ 25 & 25\end{bmatrix},$$

$$arc(6,7,5)=\begin{bmatrix}55 & 94\\ 41 & 79\\ 25 & 25\end{bmatrix}, arc(6,7,6)=\begin{bmatrix}41 & 79\\ 41 & 49\\ 25 & 25\end{bmatrix}, arc(6,7,7)=\begin{bmatrix}41 & 78\\ 41 & 49\\ 25 & 25\end{bmatrix}, arc(6,7,8)=\begin{bmatrix}41 & 78\\ 54 & 84\\ 25 & 25\end{bmatrix},$$

$$arc(6,7,9)=\begin{bmatrix}54 & 78\\ 54 & 92\\ 25 & 25\end{bmatrix}, arc(6,7,10)=\begin{bmatrix}41 & 79\\ 55 & 94\\ 25 & 25\end{bmatrix}, arc(6,7,11)=\begin{bmatrix}41 & 49\\ 41 & 79\\ 25 & 25\end{bmatrix}, arc(6,7,12)=\begin{bmatrix}41 & 49\\ 41 & 78\\ 25 & 25\end{bmatrix},$$

$$arc(6,7,13)=\begin{bmatrix}25 & 30\\ 54 & 92\\ 25 & 25\end{bmatrix}, arc(6,7,14)=\begin{bmatrix}54 & 92\\ 25 & 30\\ 25 & 25\end{bmatrix}$$

Arc [7 1]:

$$arc(7,1,1)=\begin{bmatrix}46 & 76\\ 33 & 69\\ 100 & 100\end{bmatrix}, arc(7,1,2)=\begin{bmatrix}33 & 69\\ 46 & 76\\ 100 & 100\end{bmatrix}, arc(7,1,3)=\begin{bmatrix}46 & 84\\ 46 & 69\\ 100 & 100\end{bmatrix},$$

$$arc(7,1,4)=\begin{bmatrix}46 & 69\\ 46 & 84\\ 100 & 100\end{bmatrix}, arc(7,1,5)=\begin{bmatrix}33 & 71\\ 33 & 40\\ 100 & 100\end{bmatrix}, arc(7,1,6)=\begin{bmatrix}33 & 40\\ 33 & 71\\ 100 & 100\end{bmatrix},$$

$$arc(7,1,7)=\begin{bmatrix}85 & 86\\ 31 & 70\\ 100 & 100\end{bmatrix}, arc(7,1,8)=\begin{bmatrix}31 & 70\\ 85 & 86\\ 100 & 100\end{bmatrix}, arc(7,1,9)=\begin{bmatrix}33 & 70\\ 33 & 41\\ 100 & 100\end{bmatrix},$$

$$arc(7,1,10)=\begin{bmatrix}33 & 41\\ 33 & 70\\ 100 & 100\end{bmatrix}, arc(7,1,11)=\begin{bmatrix}47 & 85\\ 33 & 71\\ 100 & 100\end{bmatrix}, arc(7,1,12)=\begin{bmatrix}33 & 71\\ 47 & 85\\ 100 & 100\end{bmatrix}$$

Arcs [7 2] and [7 4]:

$$arc(7,2)=\begin{bmatrix}17 & 22\\ 46 & 84\\ 100 & 100\end{bmatrix}, arc(7,4)=\begin{bmatrix}46 & 84\\ 17 & 22\\ 100 & 100\end{bmatrix}$$

A plot of the regions for this system appears in Figure 5.17. The shape of the regions are similar to regions of the system with idle only in Figure 5.14, but include the additional regions that skip idle. The additional idle skipping regions are concentrated around the lower threshold of a given product because a product is allowed to go from a filling node to the setup node for the next product.

Note that this system required the initial *flagged_node_list* to be set to [1 3 5 7] to get a non-oscillating set of regions for the solution. An oscillating set of regions does not change the area contained by the region set on a given arc, but is merely a different representation of the area. For example, consider a set of three regions [A B C], where A and C have no intersection but B intersects both A and C. The total region represented by the set of hyperrectangles is the union of all three regions. Note that the union of all three regions is the same as the union of A, C, and D, where $D = B - A \bigcap B - C \bigcup B$. This means that the portion of B that intersects A or C can change but it will have no effect on the representation of the set of regions. The set of regions on each arc between the setup and refill nodes ([2 3],[4 5], and [6 7]) contained the oscillating regions. Refer to Section 4.7.1 of the Stability Algorithm Chapter for further information regarding oscillating regions in the Stability Algorithm.

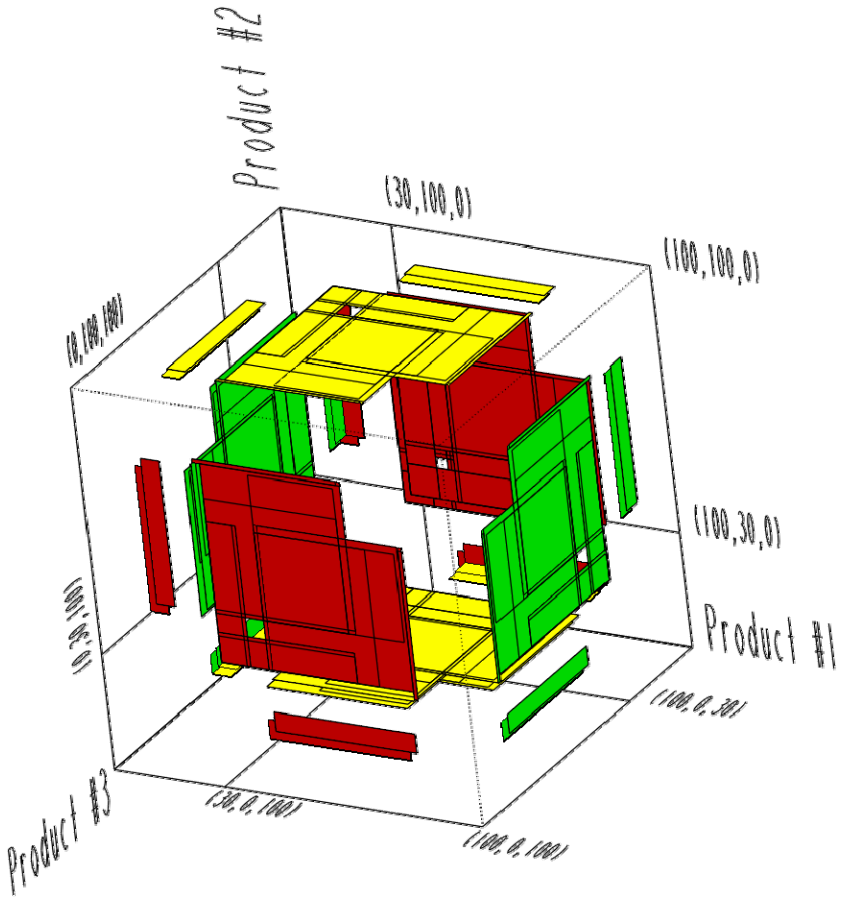


Figure 5.17: Three Product System – Skipping Idle Allowed

The results for the same system parameters but with the lower limit threshold increased to 40 units for all products will now be examined. The parameters for this system produce regions that are similar to the regions of the previous system but have smaller gaps between regions. This system provided the same non-oscillating output regardless of the initial *flagged_node_list*. The regions and plot of regions are below.

Arc [1 2]:

$$arc(1,2,1)=\begin{bmatrix}40 & 40\\ 52 & 88\\ 52 & 63\end{bmatrix}, arc(1,2,2)=\begin{bmatrix}40 & 40\\ 64 & 100\\ 52 & 88\end{bmatrix}, arc(1,2,3)=\begin{bmatrix}40 & 40\\ 52 & 63\\ 52 & 88\end{bmatrix}, arc(1,2,4)=\begin{bmatrix}40 & 40\\ 52 & 88\\ 64 & 100\end{bmatrix},$$

$$arc(1,2,5)=\begin{bmatrix}40 & 40\\ 52 & 64\\ 40 & 52\end{bmatrix}, arc(1,2,6)=\begin{bmatrix}40 & 40\\ 40 & 52\\ 52 & 64\end{bmatrix}, arc(1,2,7)=\begin{bmatrix}40 & 40\\ 66 & 100\\ 40 & 51\end{bmatrix}, arc(1,2,8)=\begin{bmatrix}40 & 40\\ 89 & 100\\ 40 & 52\end{bmatrix},$$

$$arc(1,2,9)=\begin{bmatrix}40 & 40\\ 66 & 88\\ 40 & 52\end{bmatrix}, arc(1,2,10)=\begin{bmatrix}40 & 40\\ 40 & 51\\ 66 & 100\end{bmatrix}, arc(1,2,11)=\begin{bmatrix}40 & 40\\ 40 & 52\\ 89 & 100\end{bmatrix}, arc(1,2,12)=\begin{bmatrix}40 & 40\\ 40 & 52\\ 66 & 88\end{bmatrix}$$

Arc [2 3]:

$$arc(2,3,1)=\begin{bmatrix}21 & 35\\ 95 & 95\\ 81 & 83\end{bmatrix}, arc(2,3,2)=\begin{bmatrix}21 & 35\\ 81 & 83\\ 95 & 95\end{bmatrix}, arc(2,3,3)=\begin{bmatrix}23 & 35\\ 95 & 95\\ 35 & 47\end{bmatrix}, arc(2,3,4)=\begin{bmatrix}23 & 35\\ 35 & 47\\ 95 & 95\end{bmatrix},$$

$$arc(2,3,5)=\begin{bmatrix}23 & 35\\ 95 & 95\\ 72 & 83\end{bmatrix}, arc(2,3,6)=\begin{bmatrix}23 & 34\\ 95 & 95\\ 72 & 83\end{bmatrix}, arc(2,3,7)=\begin{bmatrix}23 & 35\\ 95 & 95\\ 51 & 71\end{bmatrix}, arc(2,3,8)=\begin{bmatrix}23 & 35\\ 72 & 83\\ 95 & 95\end{bmatrix},$$

$$arc(2,3,9)=\begin{bmatrix}23 & 34\\ 51 & 83\\ 95 & 95\end{bmatrix}, arc(2,3,10)=\begin{bmatrix}23 & 35\\ 51 & 71\\ 95 & 95\end{bmatrix}, arc(2,3,11)=\begin{bmatrix}35 & 35\\ 47 & 83\\ 47 & 58\end{bmatrix}, arc(2,3,12)=\begin{bmatrix}35 & 35\\ 59 & 95\\ 47 & 83\end{bmatrix},$$

$$arc(2,3,13)=\begin{bmatrix}35 & 35\\ 47 & 59\\ 35 & 47\end{bmatrix}, arc(2,3,14)=\begin{bmatrix}35 & 35\\ 47 & 58\\ 47 & 83\end{bmatrix}, arc(2,3,15)=\begin{bmatrix}35 & 35\\ 47 & 83\\ 59 & 95\end{bmatrix}, arc(2,3,16)=\begin{bmatrix}35 & 35\\ 35 & 47\\ 47 & 59\end{bmatrix},$$

$$arc(2,3,17)=\begin{bmatrix}35 & 35\\ 61 & 95\\ 35 & 46\end{bmatrix}, arc(2,3,18)=\begin{bmatrix}35 & 35\\ 61 & 95\\ 35 & 46\end{bmatrix}, arc(2,3,19)=\begin{bmatrix}35 & 35\\ 61 & 83\\ 35 & 47\end{bmatrix}, arc(2,3,20)=\begin{bmatrix}35 & 35\\ 35 & 46\\ 61 & 95\end{bmatrix},$$

$$arc(2,3,21)=\begin{bmatrix}35 & 35\\ 35 & 47\\ 85 & 95\end{bmatrix}, arc(2,3,22)=\begin{bmatrix}35 & 35\\ 35 & 47\\ 61 & 83\end{bmatrix}$$

Arc [3 1]:

$$arc(3,1,1)=\begin{bmatrix}100 & 100\\ 40 & 76\\ 40 & 51\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 40 & 51\\ 40 & 76\end{bmatrix},$$

$$arc(3,1,3)=\begin{bmatrix}100 & 100\\ 52 & 88\\ 52 & 76\end{bmatrix}, arc(3,1,4)=\begin{bmatrix}100 & 100\\ 40 & 76\\ 52 & 88\end{bmatrix}$$

Arcs [3 4] and [3 6]:

$$arc(3,4,1)=\begin{bmatrix}100 & 100\\ 28 & 39\\ 54 & 88\end{bmatrix}, arc(3,4,2)=\begin{bmatrix}100 & 28\\ 28 & 40\\ 77 & 88\end{bmatrix}, arc(3,4,3)=\begin{bmatrix}100 & 100\\ 28 & 40\\ 72 & 83\end{bmatrix},$$

$$arc(3,4,4)=\begin{bmatrix}100 & 100\\ 28 & 40\\ 40 & 52\end{bmatrix}, arc(3,4,5)=\begin{bmatrix}100 & 100\\ 26 & 40\\ 86 & 88\end{bmatrix}, arc(3,6,1)=\begin{bmatrix}100 & 100\\ 54 & 88\\ 28 & 29\end{bmatrix},$$

$$arc(3,6,2)=\begin{bmatrix}100 & 100\\ 77 & 88\\ 28 & 40\end{bmatrix}, arc(3,6,3)=\begin{bmatrix}100 & 100\\ 54 & 76\\ 28 & 40\end{bmatrix}, arc(3,6,4)=\begin{bmatrix}100 & 100\\ 40 & 52\\ 28 & 40\end{bmatrix},$$

$$arc(3,6,5)=\begin{bmatrix}100 & 100\\ 86 & 88\\ 26 & 40\end{bmatrix}$$

Arc [1 4]:

$$arc(1,4,1)=\begin{bmatrix}52 & 88\\ 40 & 40\\ 52 & 63\end{bmatrix}, arc(1,4,2)=\begin{bmatrix}64 & 100\\ 40 & 40\\ 52 & 88\end{bmatrix}, arc(1,4,3)=\begin{bmatrix}52 & 88\\ 40 & 40\\ 64 & 100\end{bmatrix}, arc(1,4,4)=\begin{bmatrix}52 & 63\\ 40 & 40\\ 52 & 88\end{bmatrix},$$

$$arc(1,4,5)=\begin{bmatrix}66 & 100\\ 40 & 40\\ 40 & 51\end{bmatrix}, arc(1,4,6)=\begin{bmatrix}89 & 100\\ 40 & 40\\ 40 & 52\end{bmatrix}, arc(1,4,7)=\begin{bmatrix}66 & 88\\ 40 & 40\\ 40 & 52\end{bmatrix}, arc(1,4,8)=\begin{bmatrix}40 & 52\\ 40 & 40\\ 89 & 100\end{bmatrix},$$

$$arc(1,4,9)=\begin{bmatrix}40 & 51\\ 40 & 40\\ 68 & 100\end{bmatrix}, arc(1,4,10)=\begin{bmatrix}40 & 52\\ 40 & 40\\ 68 & 88\end{bmatrix}, arc(1,4,11)=\begin{bmatrix}52 & 64\\ 40 & 40\\ 40 & 52\end{bmatrix}, arc(1,4,12)=\begin{bmatrix}40 & 52\\ 40 & 40\\ 52 & 64\end{bmatrix}$$

Arc [4 5]:

$$arc(4,5,1)=\begin{bmatrix}95 & 95\\ 21 & 35\\ 81 & 83\end{bmatrix}, arc(4,5,2)=\begin{bmatrix}81 & 83\\ 21 & 35\\ 95 & 95\end{bmatrix}, arc(4,5,3)=\begin{bmatrix}95 & 95\\ 23 & 35\\ 35 & 47\end{bmatrix}, arc(4,5,4)=\begin{bmatrix}35 & 47\\ 23 & 35\\ 95 & 95\end{bmatrix},$$

$$arc(4,5,5)=\begin{bmatrix}95 & 95\\ 23 & 34\\ 49 & 83\end{bmatrix}, arc(4,5,6)=\begin{bmatrix}95 & 95\\ 23 & 35\\ 72 & 83\end{bmatrix}, arc(4,5,7)=\begin{bmatrix}95 & 95\\ 23 & 35\\ 49 & 71\end{bmatrix}, arc(4,5,8)=\begin{bmatrix}72 & 83\\ 23 & 35\\ 95 & 95\end{bmatrix},$$

$$arc(4,5,9)=\begin{bmatrix}49 & 71\\ 35 & 35\\ 63 & 95\end{bmatrix}, arc(4,5,10)=\begin{bmatrix}49 & 83\\ 23 & 34\\ 95 & 95\end{bmatrix}, arc(4,5,11)=\begin{bmatrix}47 & 83\\ 35 & 35\\ 47 & 58\end{bmatrix}, arc(4,5,12)=\begin{bmatrix}59 & 95\\ 35 & 35\\ 47 & 83\end{bmatrix},$$

$$arc(4,5,13)=\begin{bmatrix}47 & 59\\ 35 & 35\\ 35 & 47\end{bmatrix}, arc(4,5,14)=\begin{bmatrix}47 & 83\\ 35 & 35\\ 59 & 95\end{bmatrix}, arc(4,5,15)=\begin{bmatrix}47 & 58\\ 35 & 35\\ 47 & 83\end{bmatrix}, arc(4,5,16)=\begin{bmatrix}35 & 47\\ 35 & 35\\ 47 & 59\end{bmatrix},$$

$$arc(4,5,17)=\begin{bmatrix}61 & 95\\ 35 & 35\\ 35 & 46\end{bmatrix}, arc(4,5,18)=\begin{bmatrix}84 & 95\\ 35 & 35\\ 35 & 47\end{bmatrix}, arc(4,5,19)=\begin{bmatrix}61 & 83\\ 35 & 35\\ 35 & 47\end{bmatrix}, arc(4,5,20)=\begin{bmatrix}35 & 47\\ 35 & 35\\ 84 & 95\end{bmatrix},$$

$$arc(4,5,21)=\begin{bmatrix}35 & 46\\ 35 & 35\\ 63 & 95\end{bmatrix}, arc(4,5,22)=\begin{bmatrix}35 & 47\\ 35 & 35\\ 63 & 83\end{bmatrix}$$

Arc [5 1]:

$$arc(5,1,1)=\begin{bmatrix}40 & 76\\ 100 & 100\\ 40 & 51\end{bmatrix}, arc(5,1,2)=\begin{bmatrix}40 & 51\\ 100 & 100\\ 40 & 76\end{bmatrix},$$

$$arc(5,1,3)=\begin{bmatrix}52 & 88\\ 100 & 100\\ 40 & 76\end{bmatrix}, arc(5,1,4)=\begin{bmatrix}40 & 76\\ 100 & 100\\ 52 & 88\end{bmatrix}$$

Arcs [5 2] and [5 6]:

$$arc(5,2,1)=\begin{bmatrix}28 & 40\\ 100 & 100\\ 77 & 88\end{bmatrix}, arc(5,2,2)=\begin{bmatrix}28 & 39\\ 100 & 100\\ 56 & 88\end{bmatrix}, arc(5,2,3)=\begin{bmatrix}28 & 40\\ 100 & 100\\ 56 & 76\end{bmatrix},$$

$$arc(5,2,4)=\begin{bmatrix}28 & 40\\ 100 & 100\\ 40 & 52\end{bmatrix}, arc(5,2,5)=\begin{bmatrix}26 & 40\\ 100 & 100\\ 86 & 88\end{bmatrix}, arc(5,6,1)=\begin{bmatrix}54 & 88\\ 100 & 100\\ 28 & 39\end{bmatrix},$$

$$arc(5,6,2)=\begin{bmatrix}77 & 88\\ 100 & 100\\ 28 & 40\end{bmatrix}, arc(5,6,3)=\begin{bmatrix}54 & 76\\ 100 & 100\\ 28 & 40\end{bmatrix}, arc(5,6,4)=\begin{bmatrix}40 & 52\\ 100 & 100\\ 28 & 40\end{bmatrix},$$

$$arc(5,6,5)=\begin{bmatrix}86 & 88\\ 100 & 100\\ 26 & 40\end{bmatrix}$$

Arc [1 6]:

$$arc(1,6,1)=\begin{bmatrix}64 & 100\\ 52 & 88\\ 40 & 40\end{bmatrix}, arc(1,6,2)=\begin{bmatrix}52 & 88\\ 52 & 63\\ 40 & 40\end{bmatrix}, arc(1,6,3)=\begin{bmatrix}52 & 88\\ 64 & 100\\ 40 & 40\end{bmatrix}, arc(1,6,4)=\begin{bmatrix}53 & 63\\ 52 & 88\\ 40 & 40\end{bmatrix},$$

$$arc(1,6,5)=\begin{bmatrix}89 & 100\\ 40 & 52\\ 40 & 40\end{bmatrix}, arc(1,6,6)=\begin{bmatrix}66 & 100\\ 40 & 51\\ 40 & 40\end{bmatrix}, arc(1,6,7)=\begin{bmatrix}66 & 88\\ 40 & 52\\ 40 & 40\end{bmatrix}, arc(1,6,8)=\begin{bmatrix}40 & 52\\ 89 & 100\\ 40 & 40\end{bmatrix},$$

$$arc(1,6,9)=\begin{bmatrix}40 & 51\\ 68 & 100\\ 40 & 40\end{bmatrix}, arc(1,6,10)=\begin{bmatrix}40 & 52\\ 68 & 88\\ 40 & 40\end{bmatrix}, arc(1,6,11)=\begin{bmatrix}52 & 64\\ 40 & 52\\ 40 & 40\end{bmatrix}, arc(1,6,12)=\begin{bmatrix}40 & 52\\ 52 & 64\\ 40 & 40\end{bmatrix}$$

Arc [6 7]:

$$arc(6,7,1)=\begin{bmatrix}95 & 95\\ 81 & 83\\ 21 & 35\end{bmatrix}, arc(6,7,2)=\begin{bmatrix}81 & 83\\ 95 & 95\\ 21 & 35\end{bmatrix}, arc(6,7,3)=\begin{bmatrix}95 & 95\\ 35 & 47\\ 23 & 35\end{bmatrix}, arc(6,7,4)=\begin{bmatrix}35 & 47\\ 95 & 95\\ 23 & 35\end{bmatrix},$$

$$arc(6,7,5)=\begin{bmatrix}95 & 95\\ 49 & 83\\ 23 & 34\end{bmatrix}, arc(6,7,6)=\begin{bmatrix}95 & 95\\ 72 & 83\\ 23 & 35\end{bmatrix}, arc(6,7,7)=\begin{bmatrix}95 & 95\\ 49 & 71\\ 23 & 35\end{bmatrix}, arc(6,7,8)=\begin{bmatrix}72 & 83\\ 95 & 95\\ 23 & 35\end{bmatrix},$$

$$arc(6,7,9)=\begin{bmatrix}49 & 71\\ 95 & 95\\ 23 & 35\end{bmatrix}, arc(6,7,10)=\begin{bmatrix}49 & 83\\ 95 & 95\\ 23 & 34\end{bmatrix}, arc(6,7,11)=\begin{bmatrix}59 & 95\\ 47 & 83\\ 35 & 35\end{bmatrix}, arc(6,7,12)=\begin{bmatrix}47 & 83\\ 47 & 58\\ 35 & 35\end{bmatrix},$$

$$arc(6,7,13)=\begin{bmatrix}47 & 59\\ 35 & 47\\ 35 & 35\end{bmatrix}, arc(6,7,14)=\begin{bmatrix}47 & 83\\ 59 & 95\\ 35 & 35\end{bmatrix}, arc(6,7,15)=\begin{bmatrix}47 & 58\\ 47 & 83\\ 35 & 35\end{bmatrix}, arc(6,7,16)=\begin{bmatrix}35 & 47\\ 47 & 59\\ 35 & 35\end{bmatrix},$$
$$arc(6,7,17)=\begin{bmatrix}84 & 95\\ 35 & 47\\ 35 & 35\end{bmatrix}, arc(6,7,18)=\begin{bmatrix}61 & 95\\ 35 & 47\\ 35 & 35\end{bmatrix}, arc(6,7,19)=\begin{bmatrix}61 & 83\\ 35 & 47\\ 35 & 35\end{bmatrix}, arc(6,7,20)=\begin{bmatrix}35 & 47\\ 84 & 95\\ 35 & 35\end{bmatrix},$$
$$arc(6,7,21)=\begin{bmatrix}35 & 46\\ 63 & 95\\ 35 & 35\end{bmatrix}, arc(6,7,22)=\begin{bmatrix}35 & 47\\ 63 & 83\\ 35 & 35\end{bmatrix}$$

Arc [7 1]:

$$arc(7,1,1)=\begin{bmatrix}40 & 76\\ 40 & 51\\ 100 & 100\end{bmatrix}, arc(7,1,2)=\begin{bmatrix}40 & 51\\ 40 & 76\\ 100 & 100\end{bmatrix},$$
$$arc(7,1,3)=\begin{bmatrix}52 & 88\\ 40 & 76\\ 100 & 100\end{bmatrix}, arc(7,1,4)=\begin{bmatrix}40 & 76\\ 52 & 88\\ 100 & 100\end{bmatrix}$$

Arcs [7 2] and [7 4]:

$$arc(7,2,1)=\begin{bmatrix}28 & 40\\ 77 & 88\\ 100 & 100\end{bmatrix}, arc(7,2,2)=\begin{bmatrix}28 & 39\\ 56 & 88\\ 100 & 100\end{bmatrix}, arc(7,2,3)=\begin{bmatrix}28 & 40\\ 56 & 76\\ 100 & 100\end{bmatrix},$$
$$arc(7,2,4)=\begin{bmatrix}28 & 40\\ 40 & 52\\ 100 & 100\end{bmatrix}, arc(7,2,5)=\begin{bmatrix}26 & 40\\ 86 & 88\\ 100 & 100\end{bmatrix}, arc(7,4,1)=\begin{bmatrix}77 & 88\\ 28 & 40\\ 100 & 100\end{bmatrix},$$
$$arc(7,4,2)=\begin{bmatrix}54 & 88\\ 28 & 39\\ 100 & 100\end{bmatrix}, arc(7,4,3)=\begin{bmatrix}54 & 76\\ 28 & 40\\ 100 & 100\end{bmatrix}, arc(7,4,4)=\begin{bmatrix}40 & 52\\ 28 & 40\\ 100 & 100\end{bmatrix},$$
$$arc(7,4,5)=\begin{bmatrix}86 & 88\\ 26 & 40\\ 100 & 100\end{bmatrix}$$

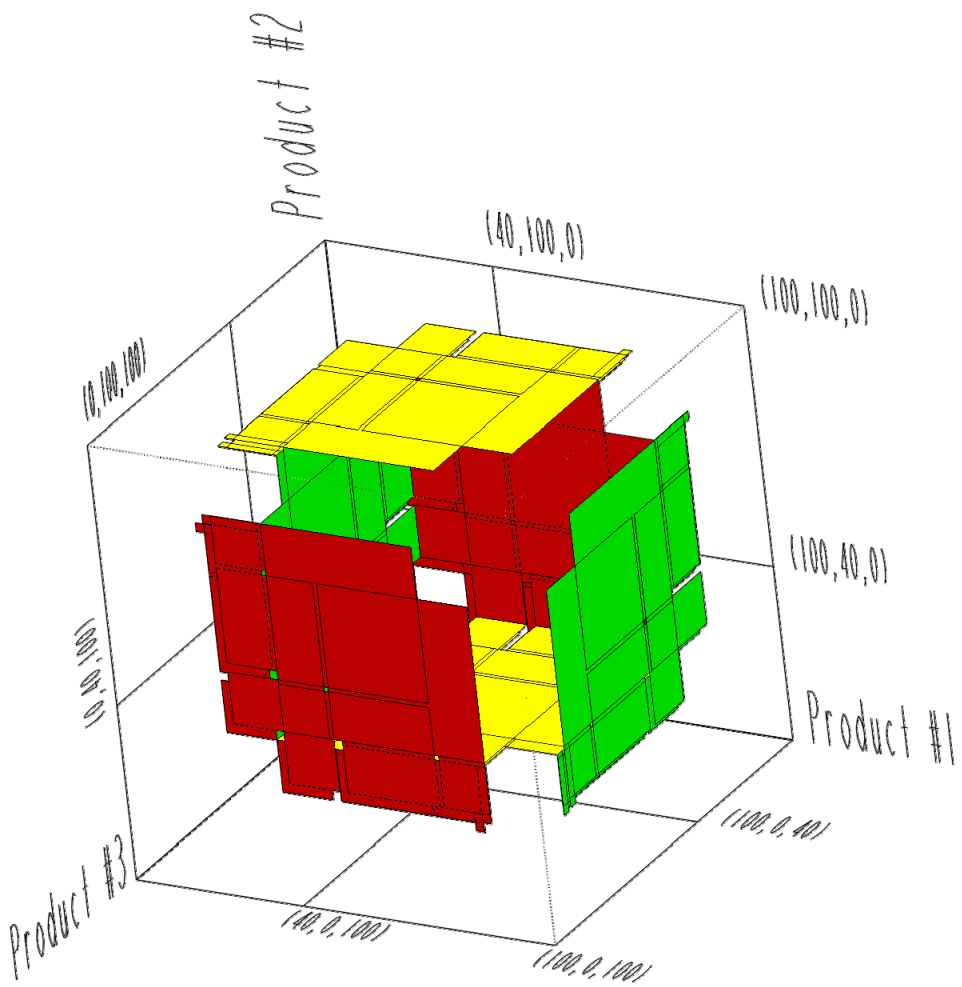


Figure 5.18: Three Product System – 40 Unit Lower Threshold

Other systems were tested that varied the lower and upper buffer thresholds and all provided similar product value regions.

#### 5.3.2.3 Three Product Network – Different Usage Rates

This section will begin to examine the effect of changing the usage rate of one product. A three product system that is non-symmetric becomes very difficult, if not impossible to evaluate using hand calculations. Consider a system with three products with the same node/arc network as the previous example and the following parameters: usage rate = 2 products/time unit for product number one, the usage rate = 1 product/time unit for products two and three, all products have the same parameters of a lower threshold = 40 products, full buffer level = 80 products, and setup time of 5 units. In this example network, all product sequences may pass through the idle node prior to being replenished or a product can enter setup directly after replenishment of the previous

product. The algorithm cycled through the network approximately seven times which took a few minutes to complete the analysis. The regions output by the algorithm are listed below and a plot of the regions follows.

Arc [1 2]:

$$arc(1,2,1)=\begin{bmatrix}40 & 40\\ 82 & 100\\ 40 & 74\end{bmatrix}, arc(1,2,2)=\begin{bmatrix}40 & 40\\ 94 & 100\\ 54 & 88\end{bmatrix}, arc(1,2,3)=\begin{bmatrix}40 & 40\\ 40 & 74\\ 82 & 100\end{bmatrix},$$

$$arc(1,2,4)=\begin{bmatrix}40 & 40\\ 54 & 88\\ 94 & 100\end{bmatrix}, arc(1,2,5)=\begin{bmatrix}40 & 40\\ 40 & 44\\ 51 & 56\end{bmatrix}, arc(1,2,6)=\begin{bmatrix}40 & 40\\ 51 & 56\\ 40 & 44\end{bmatrix}$$

Arc [2 3]:

$$arc(2,3,1)=\begin{bmatrix}6 & 30\\ 95 & 95\\ 53 & 83\end{bmatrix}, arc(2,3,2)=\begin{bmatrix}6 & 30\\ 95 & 95\\ 35 & 49\end{bmatrix}, arc(2,3,3)=\begin{bmatrix}6 & 30\\ 53 & 83\\ 95 & 95\end{bmatrix}, arc(2,3,4)=\begin{bmatrix}6 & 30\\ 35 & 49\\ 95 & 95\end{bmatrix},$$

$$arc(2,3,5)=\begin{bmatrix}30 & 30\\ 35 & 39\\ 46 & 51\end{bmatrix}, arc(2,3,6)=\begin{bmatrix}30 & 30\\ 46 & 51\\ 35 & 39\end{bmatrix}, arc(2,3,7)=\begin{bmatrix}30 & 30\\ 89 & 95\\ 49 & 83\end{bmatrix}, arc(2,3,8)=\begin{bmatrix}30 & 30\\ 49 & 83\\ 89 & 95\end{bmatrix},$$

$$arc(2,3,9)=\begin{bmatrix}30 & 30\\ 77 & 95\\ 35 & 69\end{bmatrix}, arc(2,3,10)=\begin{bmatrix}30 & 30\\ 35 & 69\\ 77 & 95\end{bmatrix}$$

Arc [3 1]:

$$arc(3,1,1)=\begin{bmatrix}100 & 100\\ 68 & 86\\ 40 & 60\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}100 & 100\\ 81 & 86\\ 40 & 60\end{bmatrix}, arc(3,1,3)=\begin{bmatrix}100 & 100\\ 40 & 60\\ 68 & 86\end{bmatrix}, arc(3,1,4)=\begin{bmatrix}100 & 100\\ 40 & 74\\ 81 & 86\end{bmatrix}$$

Arcs [3 4] and [3 6]:

$$arc(3,4,1)=\begin{bmatrix}100 & 100\\ 23 & 40\\ 83 & 86\end{bmatrix}, arc(3,4,2)=\begin{bmatrix}100 & 100\\ 26 & 30\\ 37 & 43\end{bmatrix}, arc(3,4,3)=\begin{bmatrix}100 & 100\\ 37 & 40\\ 26 & 30\end{bmatrix}, arc(3,4,4)=\begin{bmatrix}100 & 100\\ 26 & 40\\ 68 & 86\end{bmatrix}$$

$$arc(3,6,1)=\begin{bmatrix}100 & 100\\ 83 & 86\\ 23 & 40\end{bmatrix}, arc(3,6,2)=\begin{bmatrix}100 & 100\\ 26 & 30\\ 37 & 40\end{bmatrix}, arc(3,6,3)=\begin{bmatrix}100 & 100\\ 37 & 43\\ 26 & 30\end{bmatrix}, arc(3,6,4)=\begin{bmatrix}100 & 100\\ 68 & 86\\ 26 & 40\end{bmatrix}$$

Arc [1 4]:

$$arc(1,4,1)=\begin{bmatrix}64 & 100\\ 40 & 40\\ 52 & 86\end{bmatrix}, arc(1,4,2)=\begin{bmatrix}40 & 64\\ 40 & 40\\ 52 & 66\end{bmatrix}, arc(1,4,3)=\begin{bmatrix}64 & 76\\ 40 & 40\\ 66 & 100\end{bmatrix}, arc(1,4,4)=\begin{bmatrix}40 & 64\\ 40 & 40\\ 70 & 100\end{bmatrix}$$

Arc [4 5]:

$$arc(4,5,1)=\begin{bmatrix}90 & 90\\ 18 & 35\\ 78 & 81\end{bmatrix}, arc(4,5,2)=\begin{bmatrix}90 & 90\\ 21 & 25\\ 32 & 38\end{bmatrix}, arc(4,5,3)=\begin{bmatrix}90 & 90\\ 32 & 35\\ 21 & 25\end{bmatrix}, arc(4,5,4)=\begin{bmatrix}65 & 66\\ 9 & 13\\ 95 & 95\end{bmatrix},$$

$$arc(4,5,5)=\begin{bmatrix}63 & 63\\ 18 & 24\\ 95 & 95\end{bmatrix}, arc(4,5,6)=\begin{bmatrix}90 & 90\\ 21 & 35\\ 63 & 81\end{bmatrix}, arc(4,5,7)=\begin{bmatrix}54 & 90\\ 35 & 35\\ 47 & 81\end{bmatrix} arc(4,5,8)=\begin{bmatrix}30 & 54\\ 35 & 35\\ 47 & 61\end{bmatrix},$$

$$arc(4,5,9)=\begin{bmatrix}54 & 66\\ 35 & 35\\ 61 & 95\end{bmatrix}, arc(4,5,10)=\begin{bmatrix}30 & 54\\ 35 & 35\\ 65 & 95\end{bmatrix}$$

Arc [5 1]:

$$arc(5,1,1)=\begin{bmatrix}40 & 76\\ 100 & 100\\ 40 & 74\end{bmatrix}, arc(5,1,2)=\begin{bmatrix}40 & 51\\ 100 & 100\\ 54 & 88\end{bmatrix}$$

Arcs [5 2] and [5 6]:

$$arc(5,2,1)=\begin{bmatrix}16 & 40\\ 100 & 100\\ 58 & 88\end{bmatrix}, arc(5,2,2)=\begin{bmatrix}16 & 40\\ 100 & 100\\ 40 & 54\end{bmatrix} arc(5,2,3)=\begin{bmatrix}40 & 40\\ 100 & 100\\ 40 & 88\end{bmatrix}$$

$$arc(5,6,1)=\begin{bmatrix}73 & 73\\ 100 & 100\\ 23 & 29\end{bmatrix}, arc(5,6,2)=\begin{bmatrix}75 & 76\\ 100 & 100\\ 14 & 18\end{bmatrix}$$

Arc [1 6]:

$$arc(1,6,1)=\begin{bmatrix}64 & 100\\ 52 & 86\\ 40 & 40\end{bmatrix}, arc(1,6,2)=\begin{bmatrix}40 & 64\\ 52 & 66\\ 40 & 40\end{bmatrix}, arc(1,6,3)=\begin{bmatrix}64 & 76\\ 66 & 100\\ 40 & 40\end{bmatrix}, arc(1,6,4)=\begin{bmatrix}40 & 64\\ 70 & 100\\ 40 & 40\end{bmatrix}$$

Arc [6 7]:

$$arc(6,7,1)=\begin{bmatrix}90 & 90\\ 78 & 81\\ 18 & 35\end{bmatrix}, arc(6,7,2)=\begin{bmatrix}90 & 90\\ 32 & 38\\ 21 & 25\end{bmatrix}, arc(6,7,3)=\begin{bmatrix}90 & 90\\ 21 & 25\\ 32 & 35\end{bmatrix}, arc(6,7,4)=\begin{bmatrix}65 & 66\\ 95 & 95\\ 9 & 13\end{bmatrix},$$

$$arc(6,7,5)=\begin{bmatrix}63 & 63\\ 95 & 95\\ 18 & 24\end{bmatrix}, arc(6,7,6)=\begin{bmatrix}90 & 90\\ 63 & 81\\ 21 & 35\end{bmatrix}, arc(6,7,7)=\begin{bmatrix}54 & 90\\ 47 & 81\\ 35 & 35\end{bmatrix}, arc(6,7,8)=\begin{bmatrix}30 & 54\\ 47 & 61\\ 35 & 35\end{bmatrix},$$

$$arc(6,7,9)=\begin{bmatrix}54 & 66\\ 61 & 95\\ 35 & 35\end{bmatrix}, arc(6,7,10)=\begin{bmatrix}30 & 54\\ 65 & 95\\ 35 & 35\end{bmatrix}$$

Arc [7 1]:

$$arc(7,1,1)=\begin{bmatrix}40 & 76\\ 40 & 74\\ 100 & 100\end{bmatrix}, arc(7,1,2)=\begin{bmatrix}40 & 51\\ 54 & 88\\ 100 & 100\end{bmatrix}$$

Arcs [7 2] and [7 4]:

$$arc(7,2,1)=\begin{bmatrix}16 & 40\\ 58 & 88\\ 100 & 100\end{bmatrix}, arc(7,2,2)=\begin{bmatrix}16 & 40\\ 40 & 54\\ 100 & 100\end{bmatrix}, arc(7,2,3)=\begin{bmatrix}40 & 40\\ 40 & 88\\ 100 & 100\end{bmatrix}$$

$$arc(7,4,1)=\begin{bmatrix}75 & 76\\ 14 & 18\\ 100 & 100\end{bmatrix}, arc(7,4,2)=\begin{bmatrix}73 & 73\\ 23 & 29\\ 100 & 100\end{bmatrix}$$

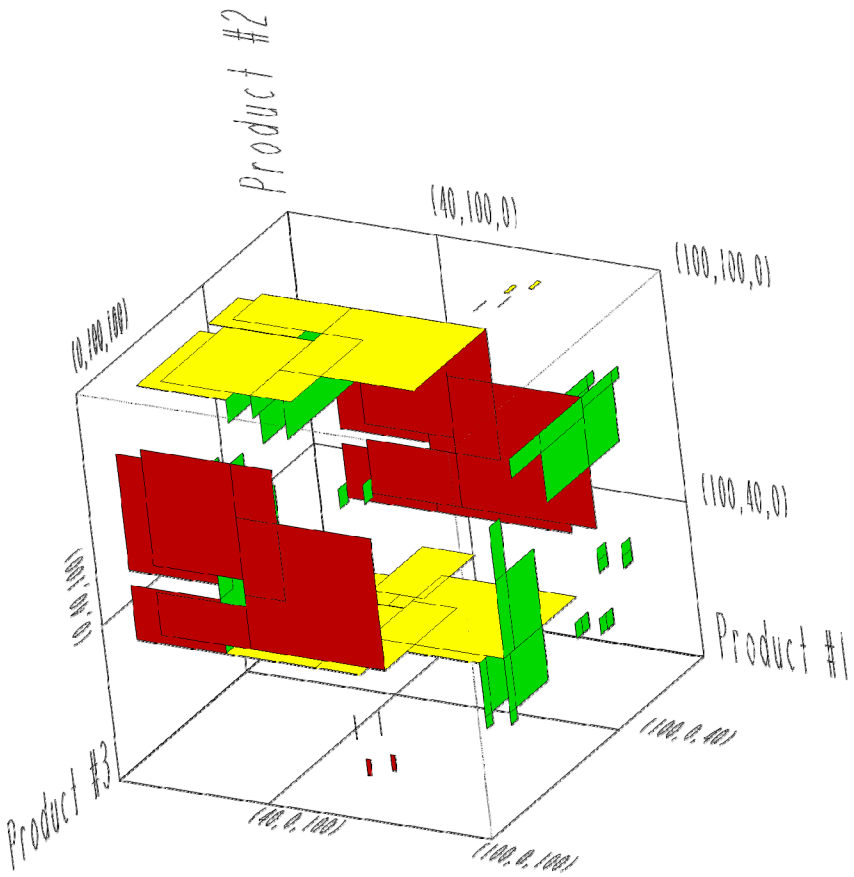


Figure 5.19: Three Product System – Differing Usage Rates

The plot of the results highlights that the differing usage rates causes the output regions to differ in shape from the previous systems as well as lose the symmetry among the products. The plot also makes it apparent that some of the regions are very small which confine the possible

combinations of product inventory levels that will be stable by mapping to an outgoing region at any given node. A larger outgoing region will allow much more freedom for various combinations of the incoming product inventory levels at a node, meaning that the incoming region has a larger target to hit when transformed by the node.

#### 5.3.2.4 Three Product System with Sequence Dependent Setups

The stability algorithm allows a user to define a wide variety of arc-node networks, to highlight the flexibility of the algorithm this example will consider a system with sequence dependent setups. Consider a three product system with the following system parameters: production rate = 10 products/time unit, usage rate = 1 product/time unit, lower threshold = 90 products, full buffer level = 150 products and idle time may or may not exist after a fill node. The setup times vary depending upon the product sequence and are as follows: product one to product two is 5 time units, product one to product three is 15 time units, product two to product one is 10 time units, product two to product three is 5 time units, product three to product one is 15 time units, and product three to product two is 15 time units. The varying setup times requires a more advance arc-node network with multiple setup nodes prior to a given filling node to capture the differing setup times. A diagram of the arc-node network is below in Figure 5.20. This system allows idle to exist after a filling node or idle can be non-existent by entering a setup node for the next product directly after refilling the previous product.

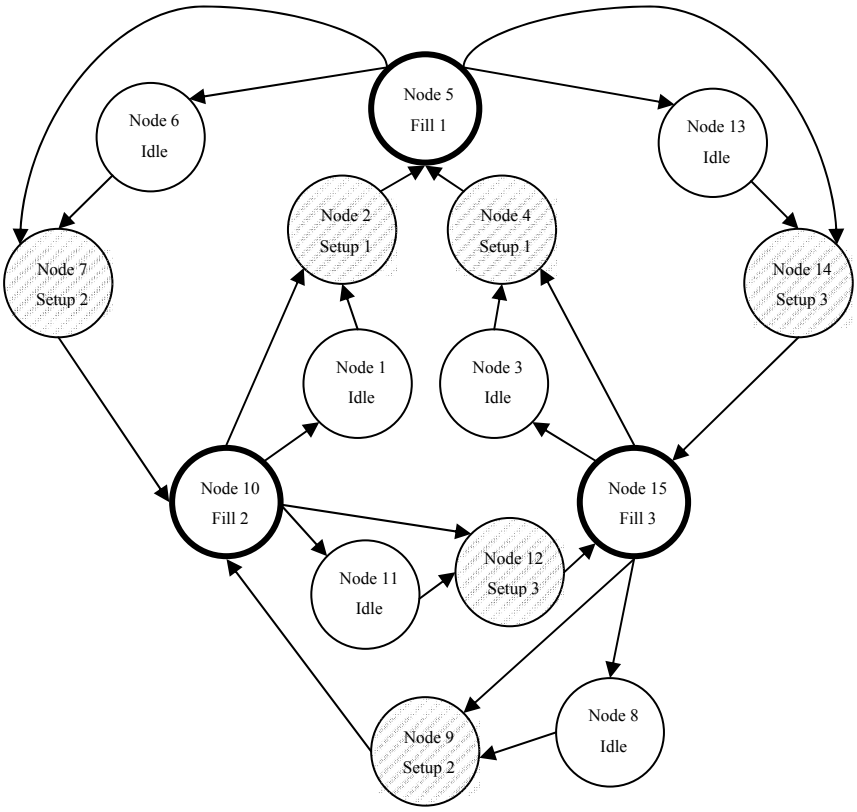


Figure 5.20: Network Map of System with Sequence Dependent Setups

The network map appears very complex, but the map is the simplest means of evaluating a sequence dependent system using the stability algorithm. This map would be very difficult to solve by hand, but the algorithm finds a solution in a few minutes as it cycles through the network 25 times. The output regions and a plot of the regions are below.

Arc [1 2]:

$$arc(1,2) = \begin{bmatrix} 90 & 90 \\ 137 & 150 \\ 108 & 127 \end{bmatrix}$$

Arc [2 5]:

$$arc(2,5,1) = \begin{bmatrix} 80 & 80 \\ 127 & 140 \\ 98 & 117 \end{bmatrix}, arc(2,5,2) = \begin{bmatrix} 74 & 80 \\ 140 & 140 \\ 98 & 98 \end{bmatrix}, arc(2,5,3) = \begin{bmatrix} 74 & 80 \\ 140 & 140 \\ 107 & 117 \end{bmatrix}$$

Arc [3 4]:

$$arc(3,4,1)=\begin{bmatrix}90 & 90\\ 90 & 91\\ 113 & 126\end{bmatrix}, arc(3,4,2)=\begin{bmatrix}90 & 90\\ 90 & 91\\ 128 & 132\end{bmatrix}, arc(3,4,3)=\begin{bmatrix}90 & 90\\ 103 & 104\\ 116 & 117\end{bmatrix},$$

$$arc(3,4,4)=\begin{bmatrix}90 & 90\\ 103 & 107\\ 117 & 119\end{bmatrix}, arc(3,4,5)=\begin{bmatrix}90 & 90\\ 103 & 104\\ 117 & 117\end{bmatrix}$$

Arc [4 5]:

$$arc(4,5,1)=\begin{bmatrix}75 & 75\\ 75 & 76\\ 98 & 111\end{bmatrix}, arc(4,5,2)=\begin{bmatrix}75 & 75\\ 88 & 92\\ 102 & 104\end{bmatrix}, arc(4,5,3)=\begin{bmatrix}75 & 75\\ 88 & 89\\ 101 & 102\end{bmatrix}$$

$$arc(4,5,4)=\begin{bmatrix}75 & 75\\ 75 & 76\\ 113 & 117\end{bmatrix}, arc(4,5,5)=\begin{bmatrix}75 & 75\\ 88 & 89\\ 102 & 102\end{bmatrix}$$

Arc [5 6]:

$$arc(5,6)=[\varnothing]$$

Arc [5 7]:

$$arc(5,7,1)=\begin{bmatrix}150 & 150\\ 67 & 68\\ 90 & 102\end{bmatrix}, arc(5,7,2)=\begin{bmatrix}150 & 150\\ 79 & 84\\ 94 & 96\end{bmatrix}, arc(5,7,3)=\begin{bmatrix}150 & 150\\ 80 & 80\\ 93 & 93\end{bmatrix}$$

$$arc(5,7,4)=\begin{bmatrix}150 & 150\\ 67 & 68\\ 105 & 109\end{bmatrix}$$

Arc [5 13]:

$$arc(5,13)=\begin{bmatrix}150 & 150\\ 119 & 132\\ 90 & 109\end{bmatrix}$$

Arc [5 14]:

$$arc(5,14,1)=\begin{bmatrix}150 & 150\\ 67 & 68\\ 90 & 90\end{bmatrix}, arc(5,14,2)=\begin{bmatrix}150 & 150\\ 132 & 132\\ 89 & 90\end{bmatrix}, arc(5,14,3)=\begin{bmatrix}150 & 150\\ 119 & 132\\ 90 & 90\end{bmatrix}$$

Arc [6 7]:

$$arc(6,7)=[\varnothing]$$

Arc [7 10]:

$$arc(7,10,1)=\begin{bmatrix}145 & 145\\ 74 & 79\\ 89 & 91\end{bmatrix}, arc(7,10,2)=\begin{bmatrix}145 & 145\\ 75 & 75\\ 88 & 88\end{bmatrix}, arc(7,10,3)=\begin{bmatrix}145 & 145\\ 62 & 63\\ 85 & 97\end{bmatrix}$$

$$arc(7,10,4)=\begin{bmatrix}145 & 145\\ 62 & 63\\ 100 & 104\end{bmatrix}$$

Arc [8 9]:

$$arc(8,9,1)=\begin{bmatrix}108 & 113\\ 90 & 90\\ 131 & 131\end{bmatrix}, arc(8,9,2)=\begin{bmatrix}108 & 113\\ 90 & 90\\ 140 & 150\end{bmatrix}, arc(8,9,3)=\begin{bmatrix}113 & 127\\ 90 & 90\\ 131 & 150\end{bmatrix}$$

Arc [9 10]:

$$arc(9,10,1)=\begin{bmatrix}112 & 112\\ 28 & 29\\ 135 & 135\end{bmatrix}, arc(9,10,2)=\begin{bmatrix}93 & 98\\ 75 & 75\\ 116 & 116\end{bmatrix}, arc(9,10,3)=\begin{bmatrix}93 & 98\\ 75 & 75\\ 125 & 135\end{bmatrix},$$

$$arc(9,10,4)=\begin{bmatrix}98 & 112\\ 75 & 75\\ 116 & 135\end{bmatrix}$$

Arc [10 1]:

$$arc(10,1)=\begin{bmatrix}90 & 103\\ 150 & 150\\ 108 & 127\end{bmatrix}$$

Arc [10 2]:

$$arc(10,2,1)=\begin{bmatrix}84 & 90\\ 150 & 150\\ 108 & 108\end{bmatrix}, arc(10,2,2)=\begin{bmatrix}84 & 90\\ 150 & 150\\ 117 & 127\end{bmatrix}, arc(10,2,3)=\begin{bmatrix}90 & 90\\ 150 & 150\\ 108 & 127\end{bmatrix}$$

Arc [10 11]:

$$arc(10,11)=\begin{bmatrix}135 & 135\\ 150 & 150\\ 90 & 94\end{bmatrix}$$

Arc [10 12]:

$$arc(10,12,1)=\begin{bmatrix}135 & 135\\ 150 & 150\\ 75 & 88\end{bmatrix}, arc(10,12,2)=\begin{bmatrix}137 & 137\\ 150 & 150\\ 80 & 83\end{bmatrix}, arc(10,12,3)=\begin{bmatrix}137 & 137\\ 150 & 150\\ 79 & 80\end{bmatrix},$$

$$arc(10,12,4)=\begin{bmatrix}137 & 137\\ 150 & 150\\ 80 & 80\end{bmatrix}$$

Arc [11 12]:

$$arc(11,12)=\begin{bmatrix}133 & 135\\ 146 & 150\\ 90 & 90\end{bmatrix}$$

Arc [12 15]:

$$arc(12,15,1)=\begin{bmatrix}130 & 130\\ 145 & 145\\ 70 & 83\end{bmatrix}, arc(12,15,2)=\begin{bmatrix}132 & 132\\ 145 & 145\\ 75 & 78\end{bmatrix}, arc(12,15,3)=\begin{bmatrix}132 & 132\\ 145 & 145\\ 74 & 75\end{bmatrix},$$

$$arc(12,15,4)=\begin{bmatrix}128 & 130\\ 141 & 145\\ 85 & 85\end{bmatrix}, arc(12,15,5)=\begin{bmatrix}132 & 132\\ 145 & 145\\ 75 & 75\end{bmatrix}$$

Arc [13 14]:

$$arc(13,14)=\begin{bmatrix}131 & 150\\ 113 & 132\\ 90 & 90\end{bmatrix}$$

Arc [14 15]:

$$arc(14,15,1)=\begin{bmatrix}135 & 135\\ 117 & 117\\ 74 & 75\end{bmatrix}, arc(14,15,2)=\begin{bmatrix}135 & 135\\ 52 & 53\\ 75 & 75\end{bmatrix}, arc(14,15,3)=\begin{bmatrix}116 & 135\\ 98 & 117\\ 75 & 75\end{bmatrix}$$

Arc [15 3]:

$$arc(15,3,1)=\begin{bmatrix}108 & 127\\ 108 & 109\\ 150 & 150\end{bmatrix}, arc(15,3,2)=\begin{bmatrix}121 & 123\\ 134 & 138\\ 150 & 150\end{bmatrix}, arc(15,3,3)=\begin{bmatrix}123 & 124\\ 137 & 137\\ 150 & 150\end{bmatrix},$$

$$arc(15,3,4)=\begin{bmatrix}123 & 123\\ 137 & 137\\ 150 & 150\end{bmatrix}$$

Arc [15 4]:

$$arc(15,4)=[\varnothing]$$

Arc [15 8]:

$$arc(15,8)=\begin{bmatrix}108 & 127\\ 90 & 109\\ 150 & 150\end{bmatrix}$$

Arc [15 9]:

$$arc(15,9,1)=\begin{bmatrix}127 & 127\\ 43 & 44\\ 150 & 150\end{bmatrix}, arc(15,9,2)=\begin{bmatrix}113 & 127\\ 90 & 90\\ 150 & 150\end{bmatrix}$$

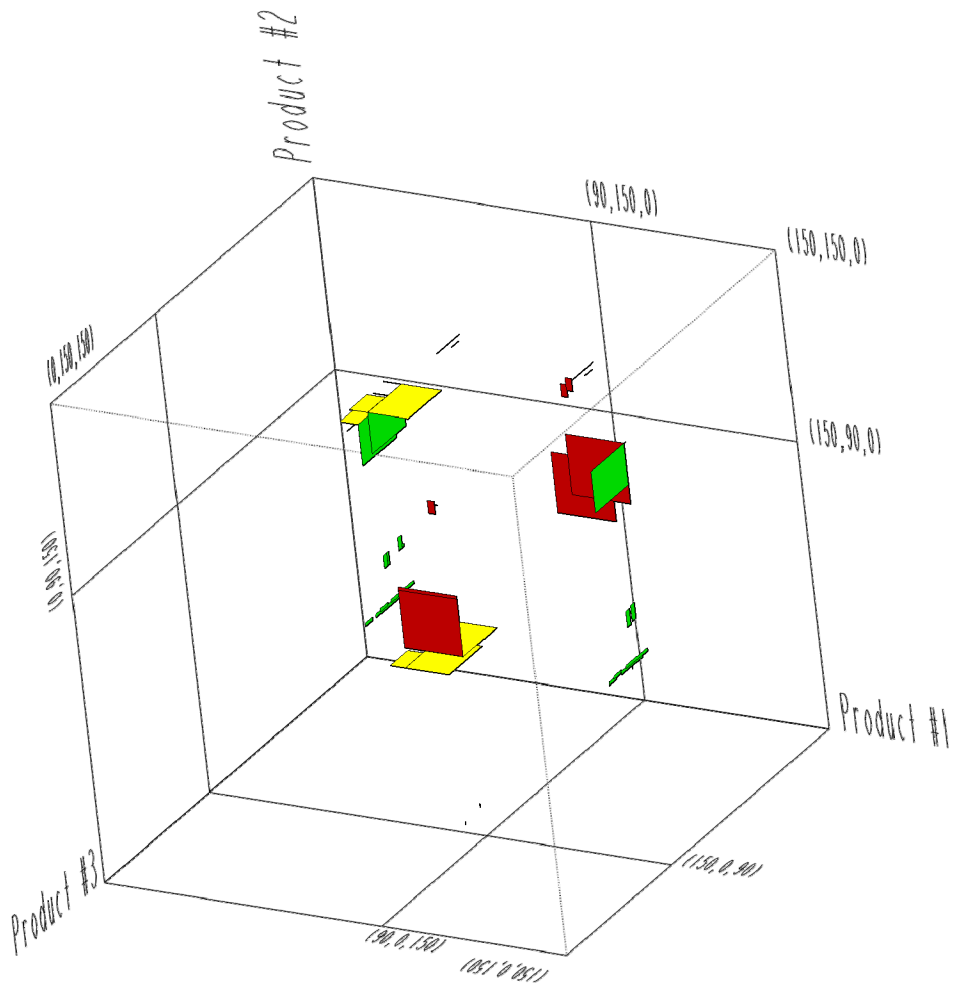


Figure 5.21: Output of Three Product System with Sequence Dependent Setups

The plot illustrates the high number of regions for this system as well as how many of the regions are very small. A region is equivalent to a target that the trajectory is aiming for, such that a large region is easier to be stable while a smaller region is potentially more difficult to hit from the previous product region.

Now consider the same arc-node network and system parameters as the previous example, but the lower threshold is decreased to 40 products and the full buffer level is decreased to 100 products. The output regions for these system parameters are below.

Arc [1 2]:

$$arc(1,2) = \begin{bmatrix} 40 & 40 \\ 87 & 100 \\ 58 & 77 \end{bmatrix}$$

Arc [2 5]:

$$arc(2,5,1)=\begin{bmatrix}30 & 30\\ 77 & 90\\ 48 & 67\end{bmatrix}, arc(2,5,2)=\begin{bmatrix}24 & 30\\ 90 & 90\\ 48 & 48\end{bmatrix}, arc(2,5,3)=\begin{bmatrix}24 & 30\\ 90 & 90\\ 51 & 67\end{bmatrix}$$

Arc [3 4]:

$$arc(3,4)=[\varnothing]$$

Arc [4 5]:

$$arc(4,5)=[\varnothing]$$

Arc [5 6]:

$$arc(5,6)=[\varnothing]$$

Arc [5 7]:

$$arc(5,7)=[\varnothing]$$

Arc [5 13]:

$$arc(5,13)=\begin{bmatrix}100 & 100\\ 69 & 82\\ 40 & 59\end{bmatrix}$$

Arc [5 14]:

$$arc(5,14,1)=\begin{bmatrix}100 & 100\\ 69 & 82\\ 40 & 40\end{bmatrix}, arc(5,14,2)=\begin{bmatrix}100 & 100\\ 82 & 82\\ 39 & 40\end{bmatrix}$$

Arc [6 7]:

$$arc(6,7)=[\varnothing]$$

Arc [7 10]:

$$arc(7,10)=[\varnothing]$$

Arc [8 9]:

$$arc(8,9,1)=\begin{bmatrix}58 & 63\\ 40 & 40\\ 81 & 81\end{bmatrix}, arc(8,9,2)=\begin{bmatrix}58 & 63\\ 40 & 40\\ 84 & 100\end{bmatrix}, arc(8,9,3)=\begin{bmatrix}63 & 77\\ 40 & 40\\ 81 & 100\end{bmatrix}$$

Arc [9 10]:

$$arc(9,10,1)=\begin{bmatrix}43 & 48\\ 25 & 25\\ 66 & 66\end{bmatrix}, arc(9,10,2)=\begin{bmatrix}43 & 48\\ 25 & 25\\ 69 & 85\end{bmatrix}, arc(9,10,3)=\begin{bmatrix}48 & 62\\ 25 & 25\\ 66 & 85\end{bmatrix}$$

Arc [10 1]:

$$arc(10,1)=\begin{bmatrix}40 & 53\\ 100 & 100\\ 58 & 77\end{bmatrix}$$

Arc [10 2]:

$$arc(10,2,1)=\begin{bmatrix}34 & 40\\ 100 & 100\\ 58 & 58\end{bmatrix}, arc(10,2,2)=\begin{bmatrix}34 & 40\\ 100 & 100\\ 61 & 77\end{bmatrix}, arc(10,2,3)=\begin{bmatrix}40 & 40\\ 100 & 100\\ 58 & 77\end{bmatrix}$$

Arc [10 11]:

$$arc(10,11)=[\varnothing]$$

Arc [10 12]:

$$arc(10,12)=[\varnothing]$$

Arc [11 12]:

$$arc(11,12)=[\varnothing]$$

Arc [12 15]:

$$arc(12,15)=[\varnothing]$$

Arc [13 14]:

$$arc(13,14)=\begin{bmatrix}81 & 100\\ 63 & 82\\ 40 & 40\end{bmatrix}$$

Arc [14 15]:

$$arc(14,15,1)=\begin{bmatrix}85 & 85\\ 67 & 67\\ 24 & 25\end{bmatrix}, arc(14,15,2)=\begin{bmatrix}66 & 85\\ 48 & 67\\ 25 & 25\end{bmatrix}$$

Arc [15 3]:

$$arc(15,3)=[\varnothing]$$

Arc [15 4]:

$$arc(15,4)=[\varnothing]$$

Arc [15 8]:

$$arc(15,8)=\begin{bmatrix}58 & 77\\ 40 & 59\\ 100 & 100\end{bmatrix}$$

Arc [15 9]:

$$arc(15,9)=\begin{bmatrix}63 & 77\\ 40 & 40\\ 100 & 100\end{bmatrix}$$

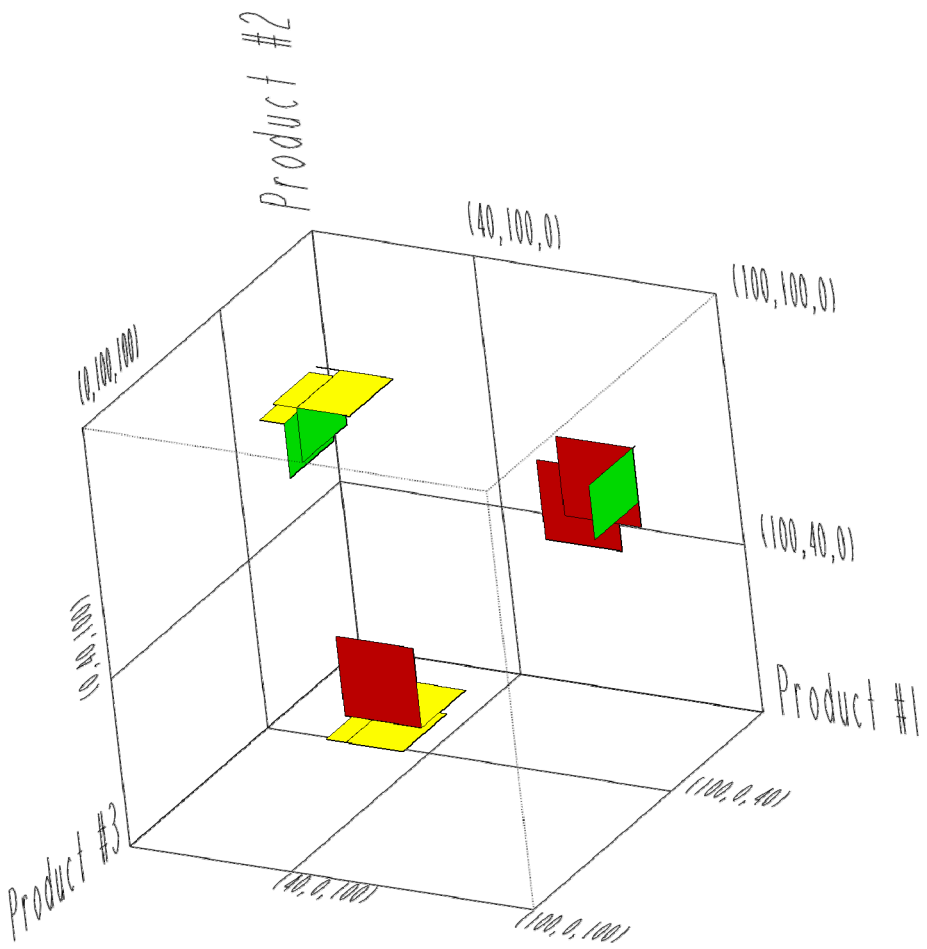


Figure 5.22: System with Sequence Dependent Setups and Smaller Buffers

The plot of the regions appears very similar to the previous example but without all of the very small regions. The arcs with the small regions in the previous example contain empty sets for this example problem. From the output of the algorithm and given the arc-node network, it is apparent that there is only one stable product sequence for this system 1-3-2. The trajectory of the sequence can either pass through idle prior to setup or the trajectory can skip idle and enter setup directly after refilling the previous product.

### 5.3.3 Four Product Network

Consider a system with four products with the node/arc network show below and the following parameters: setup time = 5 time units, production rate = 10 products/time unit, usage rate = 1 product/time unit, lower threshold = 75 products, full buffer level = 150 products. In this example network, all product sequences must pass through the idle node prior to being replenished.

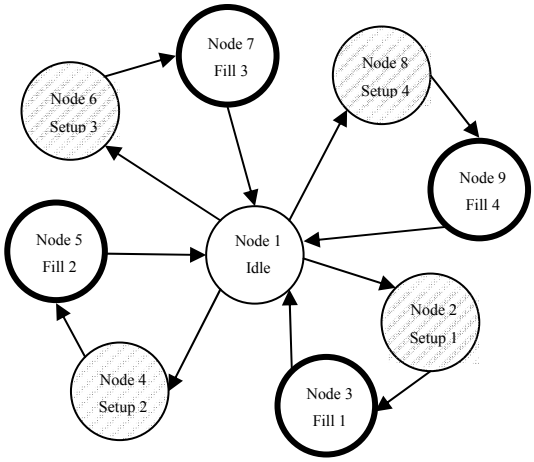


Figure 5.23: Network Map of Four-Product System – With and Without Idle

The stability algorithm required several hours to complete the analysis of this system due to the large number of arcs in and out of the idle node. Each evaluation of the idle node deals with multiple regions on the eight arcs connected to the idle node and each region is transformed, intersected and merged, all of which requires CPU time. Each time one region either into or out of the idle node is altered, the node will be flagged to be reevaluated in the future, therefore this system required hours to analyze versus minutes for the previous three product systems. The results from the algorithm are below.

Arc [1 2]:

$$arc(1,2,1)=\begin{bmatrix}75 & 75\\ 117 & 150\\ 89 & 136\\ 103 & 108\end{bmatrix}, arc(1,2,2)=\begin{bmatrix}75 & 75\\ 131 & 150\\ 89 & 136\\ 103 & 122\end{bmatrix}, arc(1,2,3)=\begin{bmatrix}75 & 75\\ 117 & 150\\ 103 & 122\\ 89 & 136\end{bmatrix},$$

$$arc(1,2,4)=\begin{bmatrix}75 & 75\\ 131 & 150\\ 103 & 122\\ 89 & 136\end{bmatrix}, arc(1,2,5)=\begin{bmatrix}75 & 75\\ 89 & 136\\ 117 & 150\\ 103 & 108\end{bmatrix}, arc(1,2,6)=\begin{bmatrix}75 & 75\\ 89 & 136\\ 131 & 150\\ 103 & 122\end{bmatrix},$$

$$arc(1,2,7)=\begin{bmatrix}75 & 75\\ 103 & 108\\ 117 & 150\\ 86 & 136\end{bmatrix}, arc(1,2,8)=\begin{bmatrix}75 & 75\\ 103 & 122\\ 131 & 150\\ 89 & 136\end{bmatrix}, arc(1,2,9)=\begin{bmatrix}75 & 75\\ 89 & 136\\ 103 & 108\\ 117 & 150\end{bmatrix},$$

$$arc(1,2,10)=\begin{bmatrix}75 & 75\\ 89 & 136\\ 103 & 122\\ 131 & 150\end{bmatrix}, arc(1,2,11)=\begin{bmatrix}75 & 75\\ 103 & 108\\ 89 & 136\\ 117 & 150\end{bmatrix}, arc(1,2,12)=\begin{bmatrix}75 & 75\\ 103 & 122\\ 89 & 136\\ 131 & 150\end{bmatrix},$$

$$arc(1,2,13)=\begin{bmatrix}75 & 75\\ 103 & 150\\ 117 & 136\\ 89 & 108\end{bmatrix}, arc(1,2,14)=\begin{bmatrix}75 & 75\\ 103 & 150\\ 89 & 108\\ 117 & 136\end{bmatrix}, arc(1,2,15)=\begin{bmatrix}75 & 75\\ 117 & 136\\ 103 & 150\\ 89 & 108\end{bmatrix},$$

$$arc(1,2,16)=\begin{bmatrix}75 & 75\\ 89 & 108\\ 103 & 150\\ 117 & 136\end{bmatrix}, arc(1,2,17)=\begin{bmatrix}75 & 75\\ 117 & 136\\ 89 & 108\\ 103 & 150\end{bmatrix}, arc(1,2,18)=\begin{bmatrix}75 & 75\\ 89 & 108\\ 117 & 136\\ 103 & 150\end{bmatrix},$$

$$arc(1,2,19)=\begin{bmatrix}75 & 75\\ 117 & 150\\ 103 & 136\\ 89 & 122\end{bmatrix}, arc(1,2,20)=\begin{bmatrix}75 & 75\\ 117 & 150\\ 89 & 122\\ 103 & 136\end{bmatrix}, arc(1,2,21)=\begin{bmatrix}75 & 75\\ 103 & 136\\ 117 & 150\\ 89 & 122\end{bmatrix},$$

$$arc(1,2,22)=\begin{bmatrix}75 & 75\\ 89 & 122\\ 117 & 150\\ 103 & 136\end{bmatrix}, arc(1,2,23)=\begin{bmatrix}75 & 75\\ 103 & 136\\ 89 & 122\\ 117 & 150\end{bmatrix}, arc(1,2,24)=\begin{bmatrix}75 & 75\\ 89 & 122\\ 103 & 136\\ 117 & 150\end{bmatrix}$$

Arc [2 3]:

$$arc(2,3,1)=\begin{bmatrix}70 & 70\\ 112 & 145\\ 84 & 131\\ 98 & 103\end{bmatrix}, arc(2,3,2)=\begin{bmatrix}70 & 70\\ 126 & 145\\ 84 & 131\\ 98 & 117\end{bmatrix}, arc(2,3,3)=\begin{bmatrix}70 & 70\\ 98 & 145\\ 84 & 103\\ 112 & 131\end{bmatrix},$$

$$arc(2,3,4)=\begin{bmatrix}70 & 70\\ 112 & 145\\ 84 & 103\\ 112 & 131\end{bmatrix}, arc(2,3,5)=\begin{bmatrix}70 & 70\\ 98 & 145\\ 112 & 131\\ 98 & 117\end{bmatrix}, arc(2,3,6)=\begin{bmatrix}70 & 70\\ 112 & 145\\ 98 & 131\\ 84 & 117\end{bmatrix},$$

$$arc(2,3,7)=\begin{bmatrix}70 & 70\\ 112 & 145\\ 98 & 103\\ 84 & 131\end{bmatrix}, arc(2,3,8)=\begin{bmatrix}70 & 70\\ 126 & 145\\ 98 & 117\\ 84 & 131\end{bmatrix}, arc(2,3,9)=\begin{bmatrix}70 & 70\\ 84 & 131\\ 112 & 145\\ 98 & 103\end{bmatrix},$$

$$arc(2,3,10)=\begin{bmatrix}70 & 70\\ 84 & 131\\ 126 & 145\\ 98 & 117\end{bmatrix}, arc(2,3,11)=\begin{bmatrix}70 & 70\\ 84 & 103\\ 98 & 145\\ 112 & 131\end{bmatrix}, arc(2,3,12)=\begin{bmatrix}70 & 70\\ 84 & 117\\ 112 & 145\\ 98 & 131\end{bmatrix},$$

$$arc(2,3,13)=\begin{bmatrix}70 & 70\\ 98 & 131\\ 112 & 145\\ 84 & 117\end{bmatrix}, arc(2,3,14)=\begin{bmatrix}70 & 70\\ 112 & 131\\ 98 & 145\\ 84 & 103\end{bmatrix}, arc(2,3,15)=\begin{bmatrix}70 & 70\\ 98 & 103\\ 112 & 145\\ 84 & 131\end{bmatrix},$$

$$arc(2,3,16)=\begin{bmatrix}70 & 70\\ 98 & 117\\ 126 & 145\\ 84 & 131\end{bmatrix}, arc(2,3,17)=\begin{bmatrix}70 & 70\\ 84 & 131\\ 98 & 103\\ 112 & 145\end{bmatrix}, arc(2,3,18)=\begin{bmatrix}70 & 70\\ 84 & 131\\ 98 & 117\\ 126 & 145\end{bmatrix},$$

$$arc(2,3,19)=\begin{bmatrix}70 & 70\\ 84 & 117\\ 98 & 131\\ 112 & 145\end{bmatrix}, arc(2,3,20)=\begin{bmatrix}70 & 70\\ 84 & 103\\ 112 & 131\\ 98 & 145\end{bmatrix}, arc(2,3,21)=\begin{bmatrix}70 & 70\\ 98 & 131\\ 84 & 117\\ 112 & 145\end{bmatrix},$$

$$arc(2,3,22)=\begin{bmatrix}70 & 70\\ 112 & 131\\ 84 & 103\\ 98 & 145\end{bmatrix}, arc(2,3,23)=\begin{bmatrix}70 & 70\\ 98 & 103\\ 84 & 131\\ 112 & 145\end{bmatrix}, arc(2,3,24)=\begin{bmatrix}70 & 70\\ 98 & 117\\ 84 & 131\\ 126 & 145\end{bmatrix}$$

Arc [3 1]:

$$arc(3,1,1)=\begin{bmatrix}150 & 150\\ 103 & 136\\ 75 & 122\\ 89 & 94\end{bmatrix}, arc(3,1,2)=\begin{bmatrix}150 & 150\\ 117 & 136\\ 75 & 122\\ 89 & 108\end{bmatrix}, arc(3,1,3)=\begin{bmatrix}150 & 150\\ 103 & 136\\ 89 & 94\\ 75 & 122\end{bmatrix},$$

$$arc(3,1,4)=\begin{bmatrix}150 & 150\\ 117 & 136\\ 89 & 108\\ 75 & 122\end{bmatrix}, arc(3,1,5)=\begin{bmatrix}150 & 150\\ 75 & 122\\ 103 & 136\\ 89 & 94\end{bmatrix}, arc(3,1,6)=\begin{bmatrix}150 & 150\\ 75 & 122\\ 117 & 136\\ 89 & 108\end{bmatrix},$$

$$arc(3,1,7)=\begin{bmatrix}150 & 150\\ 89 & 94\\ 103 & 136\\ 75 & 122\end{bmatrix}, arc(3,1,8)=\begin{bmatrix}150 & 150\\ 89 & 108\\ 117 & 136\\ 75 & 122\end{bmatrix}, arc(3,1,9)=\begin{bmatrix}150 & 150\\ 75 & 122\\ 89 & 94\\ 103 & 136\end{bmatrix},$$

$$arc(3,1,10)=\begin{bmatrix}150 & 150\\ 75 & 122\\ 89 & 108\\ 117 & 136\end{bmatrix}, arc(3,1,11)=\begin{bmatrix}150 & 150\\ 89 & 94\\ 75 & 122\\ 103 & 136\end{bmatrix}, arc(3,1,12)=\begin{bmatrix}150 & 150\\ 89 & 108\\ 75 & 122\\ 117 & 136\end{bmatrix},$$

$$arc(3,1,13)=\begin{bmatrix}150 & 150\\ 89 & 136\\ 75 & 94\\ 103 & 122\end{bmatrix}, arc(3,1,14)=\begin{bmatrix}150 & 150\\ 103 & 136\\ 75 & 108\\ 89 & 122\end{bmatrix}, arc(3,1,15)=\begin{bmatrix}150 & 150\\ 89 & 136\\ 103 & 122\\ 75 & 94\end{bmatrix},$$

$$arc(3,1,16)=\begin{bmatrix}150 & 150\\ 103 & 136\\ 89 & 122\\ 75 & 108\end{bmatrix}, arc(3,1,17)=\begin{bmatrix}150 & 150\\ 75 & 94\\ 89 & 136\\ 103 & 122\end{bmatrix}, arc(3,1,18)=\begin{bmatrix}150 & 150\\ 75 & 108\\ 103 & 136\\ 89 & 122\end{bmatrix},$$

$$arc(3,1,19)=\begin{bmatrix}150 & 150\\ 89 & 122\\ 103 & 136\\ 75 & 108\end{bmatrix}, arc(3,1,20)=\begin{bmatrix}150 & 150\\ 103 & 122\\ 89 & 136\\ 75 & 94\end{bmatrix}, arc(3,1,21)=\begin{bmatrix}150 & 150\\ 75 & 108\\ 89 & 122\\ 103 & 136\end{bmatrix},$$

$$arc(3,1,22)=\begin{bmatrix}150 & 150\\ 75 & 94\\ 103 & 122\\ 89 & 136\end{bmatrix}, arc(3,1,23)=\begin{bmatrix}150 & 150\\ 89 & 122\\ 75 & 108\\ 103 & 136\end{bmatrix}, arc(3,1,24)=\begin{bmatrix}150 & 150\\ 103 & 122\\ 75 & 94\\ 89 & 136\end{bmatrix}$$

Arc [1 4]:

$$arc(1,4,1)=\begin{bmatrix}117 & 150\\ 75 & 75\\ 89 & 136\\ 103 & 108\end{bmatrix}, arc(1,4,2)=\begin{bmatrix}131 & 150\\ 75 & 75\\ 89 & 136\\ 103 & 122\end{bmatrix}, arc(1,4,3)=\begin{bmatrix}117 & 150\\ 75 & 75\\ 103 & 108\\ 89 & 136\end{bmatrix},$$

$$arc(1,4,4)=\begin{bmatrix}131 & 150\\ 75 & 75\\ 103 & 122\\ 89 & 136\end{bmatrix}, arc(1,4,5)=\begin{bmatrix}89 & 136\\ 75 & 75\\ 117 & 150\\ 103 & 108\end{bmatrix}, arc(1,4,6)=\begin{bmatrix}89 & 136\\ 75 & 75\\ 131 & 150\\ 103 & 122\end{bmatrix},$$

$$arc(1,4,7)=\begin{bmatrix}89 & 136\\ 75 & 75\\ 103 & 108\\ 117 & 150\end{bmatrix}, arc(1,4,8)=\begin{bmatrix}89 & 136\\ 75 & 75\\ 103 & 122\\ 131 & 150\end{bmatrix}, arc(1,4,9)=\begin{bmatrix}103 & 122\\ 75 & 75\\ 131 & 150\\ 89 & 136\end{bmatrix},$$

$$arc(1,4,10)=\begin{bmatrix}103 & 122\\ 75 & 75\\ 89 & 136\\ 131 & 150\end{bmatrix}, arc(1,4,11)=\begin{bmatrix}89 & 122\\ 75 & 75\\ 117 & 150\\ 103 & 136\end{bmatrix}, arc(1,4,12)=\begin{bmatrix}103 & 136\\ 75 & 75\\ 117 & 150\\ 103 & 122\end{bmatrix},$$

$$arc(1,4,13)=\begin{bmatrix}117 & 136\\ 75 & 75\\ 103 & 150\\ 89 & 108\end{bmatrix}, arc(1,4,14)=\begin{bmatrix}89 & 122\\ 75 & 75\\ 103 & 136\\ 117 & 150\end{bmatrix}, arc(1,4,15)=\begin{bmatrix}103 & 136\\ 75 & 75\\ 89 & 122\\ 117 & 150\end{bmatrix},$$

$$arc(1,4,16)=\begin{bmatrix}117 & 136\\ 75 & 75\\ 89 & 108\\ 103 & 150\end{bmatrix}, arc(1,4,17)=\begin{bmatrix}103 & 108\\ 75 & 75\\ 117 & 150\\ 89 & 136\end{bmatrix}, arc(1,4,18)=\begin{bmatrix}103 & 108\\ 75 & 75\\ 89 & 136\\ 117 & 150\end{bmatrix},$$

$$arc(1,4,19)=\begin{bmatrix}103 & 150\\ 75 & 75\\ 117 & 136\\ 89 & 108\end{bmatrix}, arc(1,4,20)=\begin{bmatrix}117 & 150\\ 75 & 75\\ 103 & 136\\ 89 & 122\end{bmatrix}, arc(1,4,21)=\begin{bmatrix}103 & 150\\ 75 & 75\\ 89 & 108\\ 117 & 136\end{bmatrix},$$

$$arc(1,4,22)=\begin{bmatrix}117 & 150\\ 75 & 75\\ 89 & 122\\ 103 & 136\end{bmatrix}, arc(1,4,23)=\begin{bmatrix}89 & 108\\ 75 & 75\\ 103 & 150\\ 117 & 136\end{bmatrix}, arc(1,4,24)=\begin{bmatrix}89 & 108\\ 75 & 75\\ 117 & 136\\ 103 & 150\end{bmatrix}$$

Arc [4 5]:

$$arc(4,5,1)=\begin{bmatrix}112 & 145\\ 70 & 70\\ 84 & 131\\ 98 & 103\end{bmatrix}, arc(4,5,2)=\begin{bmatrix}126 & 145\\ 70 & 70\\ 84 & 131\\ 98 & 117\end{bmatrix}, arc(4,5,3)=\begin{bmatrix}98 & 145\\ 70 & 70\\ 84 & 103\\ 112 & 131\end{bmatrix},$$

$$arc(4,5,4)=\begin{bmatrix}112 & 145\\ 70 & 70\\ 84 & 117\\ 98 & 131\end{bmatrix}, arc(4,5,5)=\begin{bmatrix}98 & 145\\ 70 & 70\\ 112 & 131\\ 84 & 103\end{bmatrix}, arc(4,5,6)=\begin{bmatrix}112 & 145\\ 70 & 70\\ 98 & 131\\ 84 & 117\end{bmatrix},$$

$$arc(4,5,7)=\begin{bmatrix}112 & 145\\ 70 & 70\\ 98 & 103\\ 84 & 131\end{bmatrix}, arc(4,5,8)=\begin{bmatrix}126 & 145\\ 70 & 70\\ 98 & 117\\ 84 & 131\end{bmatrix}, arc(4,5,9)=\begin{bmatrix}84 & 131\\ 70 & 70\\ 112 & 145\\ 98 & 103\end{bmatrix},$$

$$arc(4,5,10)=\begin{bmatrix}84 & 131\\ 70 & 70\\ 126 & 145\\ 98 & 117\end{bmatrix}, arc(4,5,11)=\begin{bmatrix}84 & 117\\ 70 & 70\\ 112 & 145\\ 98 & 131\end{bmatrix}, arc(4,5,12)=\begin{bmatrix}84 & 103\\ 70 & 70\\ 98 & 145\\ 98 & 131\end{bmatrix},$$

$$arc(4,5,13)=\begin{bmatrix}98 & 131\\ 70 & 70\\ 112 & 145\\ 84 & 117\end{bmatrix}, arc(4,5,14)=\begin{bmatrix}98 & 117\\ 70 & 70\\ 126 & 145\\ 84 & 131\end{bmatrix}, arc(4,5,15)=\begin{bmatrix}98 & 103\\ 70 & 70\\ 112 & 145\\ 84 & 131\end{bmatrix},$$

$$arc(4,5,16)=\begin{bmatrix}112 & 131\\ 70 & 70\\ 98 & 145\\ 84 & 103\end{bmatrix}, arc(4,5,17)=\begin{bmatrix}84 & 131\\ 70 & 70\\ 98 & 103\\ 112 & 145\end{bmatrix}, arc(4,5,18)=\begin{bmatrix}84 & 131\\ 70 & 70\\ 98 & 117\\ 126 & 145\end{bmatrix},$$

$$arc(4,5,19)=\begin{bmatrix}84 & 103\\ 70 & 70\\ 112 & 131\\ 98 & 145\end{bmatrix}, arc(4,5,20)=\begin{bmatrix}84 & 117\\ 70 & 70\\ 98 & 131\\ 112 & 145\end{bmatrix}, arc(4,5,21)=\begin{bmatrix}98 & 131\\ 70 & 70\\ 84 & 117\\ 112 & 145\end{bmatrix},$$

$$arc(4,5,22)=\begin{bmatrix}98 & 117\\ 70 & 70\\ 84 & 131\\ 126 & 145\end{bmatrix}, arc(4,5,23)=\begin{bmatrix}98 & 103\\ 70 & 70\\ 84 & 131\\ 112 & 145\end{bmatrix}, arc(4,5,24)=\begin{bmatrix}112 & 131\\ 70 & 70\\ 84 & 103\\ 98 & 145\end{bmatrix}$$

Arc [5 1]:

$$arc(5,1,1)=\begin{bmatrix}103 & 136\\ 150 & 150\\ 75 & 122\\ 89 & 94\end{bmatrix}, arc(5,1,3)=\begin{bmatrix}117 & 136\\ 150 & 150\\ 75 & 122\\ 89 & 108\end{bmatrix}, arc(5,1,4)=\begin{bmatrix}103 & 136\\ 150 & 150\\ 89 & 94\\ 75 & 122\end{bmatrix},$$

$$arc(5,1,4)=\begin{bmatrix}117 & 136\\ 150 & 150\\ 89 & 108\\ 75 & 122\end{bmatrix}, arc(5,1,5)=\begin{bmatrix}89 & 94\\ 150 & 150\\ 75 & 122\\ 103 & 136\end{bmatrix}, arc(5,1,6)=\begin{bmatrix}89 & 94\\ 150 & 150\\ 103 & 136\\ 75 & 122\end{bmatrix},$$

$$arc(5,1,7)=\begin{bmatrix}75 & 122\\ 150 & 150\\ 103 & 136\\ 89 & 94\end{bmatrix}, arc(5,1,8)=\begin{bmatrix}75 & 122\\ 150 & 150\\ 117 & 136\\ 89 & 108\end{bmatrix}, arc(5,1,9)=\begin{bmatrix}75 & 122\\ 150 & 150\\ 89 & 94\\ 103 & 136\end{bmatrix},$$

$$arc(5,1,10)=\begin{bmatrix}75 & 122\\ 150 & 150\\ 89 & 108\\ 117 & 136\end{bmatrix}, arc(5,1,11)=\begin{bmatrix}89 & 108\\ 150 & 150\\ 75 & 122\\ 117 & 136\end{bmatrix}, arc(5,1,12)=\begin{bmatrix}89 & 108\\ 150 & 150\\ 117 & 136\\ 75 & 122\end{bmatrix},$$

$$arc(5,1,13)=\begin{bmatrix}75 & 108\\ 150 & 150\\ 103 & 136\\ 89 & 122\end{bmatrix}, arc(5,1,14)=\begin{bmatrix}75 & 94\\ 150 & 150\\ 89 & 136\\ 103 & 122\end{bmatrix}, arc(5,1,15)=\begin{bmatrix}75 & 94\\ 150 & 150\\ 103 & 122\\ 89 & 136\end{bmatrix},$$

$$arc(5,1,16)=\begin{bmatrix}75 & 108\\ 150 & 150\\ 89 & 122\\ 103 & 136\end{bmatrix}, arc(5,1,17)=\begin{bmatrix}103 & 136\\ 150 & 150\\ 75 & 108\\ 89 & 122\end{bmatrix}, arc(5,1,18)=\begin{bmatrix}89 & 122\\ 150 & 150\\ 75 & 108\\ 103 & 136\end{bmatrix},$$

$$arc(5,1,19)=\begin{bmatrix}103 & 136\\ 150 & 150\\ 89 & 122\\ 75 & 108\end{bmatrix}, arc(5,1,20)=\begin{bmatrix}89 & 122\\ 150 & 150\\ 103 & 136\\ 75 & 108\end{bmatrix}, arc(5,1,21)=\begin{bmatrix}103 & 122\\ 150 & 150\\ 75 & 94\\ 89 & 136\end{bmatrix},$$

$$arc(5,1,22)=\begin{bmatrix}103 & 122\\ 150 & 150\\ 89 & 136\\ 75 & 94\end{bmatrix}, arc(5,1,23)=\begin{bmatrix}89 & 136\\ 150 & 150\\ 75 & 94\\ 103 & 122\end{bmatrix}, arc(5,1,24)=\begin{bmatrix}89 & 136\\ 150 & 150\\ 103 & 122\\ 75 & 94\end{bmatrix}$$

Arc [1 6]:

$$arc(1,6,1)=\begin{bmatrix}117 & 150\\ 89 & 136\\ 75 & 75\\ 103 & 108\end{bmatrix}, arc(1,6,2)=\begin{bmatrix}131 & 150\\ 89 & 136\\ 75 & 75\\ 103 & 122\end{bmatrix}, arc(1,6,3)=\begin{bmatrix}117 & 150\\ 103 & 108\\ 75 & 75\\ 89 & 136\end{bmatrix},$$

$$arc(1,6,4)=\begin{bmatrix}131 & 150\\ 103 & 122\\ 75 & 75\\ 89 & 136\end{bmatrix}, arc(1,6,5)=\begin{bmatrix}89 & 136\\ 117 & 150\\ 75 & 75\\ 103 & 108\end{bmatrix}, arc(1,6,6)=\begin{bmatrix}89 & 136\\ 131 & 150\\ 75 & 75\\ 103 & 122\end{bmatrix},$$

$$arc(1,6,7)=\begin{bmatrix}89 & 136\\ 103 & 108\\ 75 & 75\\ 117 & 150\end{bmatrix}, arc(1,6,8)=\begin{bmatrix}89 & 136\\ 103 & 122\\ 75 & 75\\ 131 & 150\end{bmatrix}, arc(1,6,9)=\begin{bmatrix}103 & 122\\ 131 & 150\\ 75 & 75\\ 89 & 136\end{bmatrix},$$

$$arc(1,6,10)=\begin{bmatrix}103 & 122\\ 89 & 136\\ 75 & 75\\ 131 & 150\end{bmatrix}, arc(1,6,11)=\begin{bmatrix}89 & 122\\ 117 & 150\\ 75 & 75\\ 103 & 136\end{bmatrix}, arc(1,6,12)=\begin{bmatrix}103 & 136\\ 117 & 150\\ 75 & 75\\ 89 & 122\end{bmatrix},$$

$$arc(1,6,13)=\begin{bmatrix}117 & 136\\ 103 & 150\\ 75 & 75\\ 89 & 108\end{bmatrix}, arc(1,6,14)=\begin{bmatrix}89 & 122\\ 103 & 136\\ 75 & 75\\ 117 & 150\end{bmatrix}, arc(1,6,15)=\begin{bmatrix}103 & 136\\ 89 & 122\\ 75 & 75\\ 117 & 150\end{bmatrix},$$

$$arc(1,6,16)=\begin{bmatrix}117 & 136\\ 89 & 108\\ 75 & 75\\ 103 & 150\end{bmatrix}, arc(1,6,17)=\begin{bmatrix}103 & 108\\ 117 & 150\\ 75 & 75\\ 89 & 136\end{bmatrix}, arc(1,6,18)=\begin{bmatrix}103 & 108\\ 89 & 136\\ 75 & 75\\ 117 & 150\end{bmatrix},$$

$$arc(1,6,19)=\begin{bmatrix}103 & 150\\ 117 & 136\\ 75 & 75\\ 89 & 108\end{bmatrix}, arc(1,6,20)=\begin{bmatrix}117 & 150\\ 103 & 136\\ 75 & 75\\ 89 & 122\end{bmatrix}, arc(1,6,21)=\begin{bmatrix}103 & 150\\ 89 & 108\\ 75 & 75\\ 117 & 136\end{bmatrix},$$

$$arc(1,6,22)=\begin{bmatrix}117 & 150\\ 89 & 122\\ 75 & 75\\ 103 & 136\end{bmatrix}, arc(1,6,23)=\begin{bmatrix}89 & 108\\ 103 & 150\\ 75 & 75\\ 117 & 136\end{bmatrix}, arc(1,6,24)=\begin{bmatrix}89 & 108\\ 117 & 136\\ 75 & 75\\ 103 & 150\end{bmatrix}$$

Arc [6 7]:

$$arc(6,7,1)=\begin{bmatrix}112 & 145\\ 84 & 131\\ 70 & 70\\ 98 & 103\end{bmatrix}, arc(6,7,2)=\begin{bmatrix}126 & 145\\ 84 & 131\\ 70 & 70\\ 98 & 117\end{bmatrix}, arc(6,7,3)=\begin{bmatrix}98 & 145\\ 84 & 103\\ 70 & 70\\ 112 & 131\end{bmatrix},$$

$$arc(6,7,4)=\begin{bmatrix}112 & 145\\ 84 & 117\\ 70 & 70\\ 98 & 131\end{bmatrix}, arc(6,7,5)=\begin{bmatrix}98 & 145\\ 112 & 131\\ 70 & 70\\ 84 & 103\end{bmatrix}, arc(6,7,6)=\begin{bmatrix}112 & 145\\ 98 & 131\\ 70 & 70\\ 84 & 117\end{bmatrix},$$

$$arc(6,7,7)=\begin{bmatrix}112 & 145\\ 98 & 103\\ 70 & 70\\ 84 & 131\end{bmatrix}, arc(6,7,8)=\begin{bmatrix}126 & 145\\ 98 & 117\\ 70 & 70\\ 84 & 131\end{bmatrix}, arc(6,7,9)=\begin{bmatrix}84 & 131\\ 112 & 145\\ 70 & 70\\ 98 & 103\end{bmatrix},$$

$$arc(6,7,10)=\begin{bmatrix}84 & 131\\ 126 & 145\\ 70 & 70\\ 98 & 117\end{bmatrix}, arc(6,7,11)=\begin{bmatrix}84 & 117\\ 112 & 145\\ 70 & 70\\ 98 & 131\end{bmatrix}, arc(6,7,12)=\begin{bmatrix}84 & 103\\ 98 & 145\\ 70 & 70\\ 112 & 131\end{bmatrix},$$

$$arc(6,7,13)=\begin{bmatrix}98 & 131\\ 112 & 145\\ 70 & 70\\ 84 & 117\end{bmatrix}, arc(6,7,14)=\begin{bmatrix}98 & 117\\ 126 & 145\\ 70 & 70\\ 84 & 131\end{bmatrix}, arc(6,7,15)=\begin{bmatrix}98 & 103\\ 112 & 145\\ 70 & 70\\ 84 & 131\end{bmatrix},$$

$$arc(6,7,16)=\begin{bmatrix}112 & 131\\ 98 & 145\\ 70 & 70\\ 84 & 103\end{bmatrix}, arc(6,7,17)=\begin{bmatrix}84 & 131\\ 98 & 103\\ 70 & 70\\ 112 & 145\end{bmatrix}, arc(6,7,18)=\begin{bmatrix}84 & 131\\ 98 & 117\\ 70 & 70\\ 126 & 145\end{bmatrix},$$

$$arc(6,7,19)=\begin{bmatrix}84 & 103\\ 112 & 131\\ 70 & 70\\ 98 & 145\end{bmatrix}, arc(6,7,20)=\begin{bmatrix}84 & 117\\ 98 & 131\\ 70 & 70\\ 112 & 145\end{bmatrix}, arc(6,7,21)=\begin{bmatrix}98 & 131\\ 84 & 117\\ 70 & 70\\ 112 & 145\end{bmatrix},$$

$$arc(6,7,22)=\begin{bmatrix}98 & 117\\ 84 & 131\\ 70 & 70\\ 126 & 145\end{bmatrix}, arc(6,7,23)=\begin{bmatrix}98 & 103\\ 84 & 131\\ 70 & 70\\ 112 & 145\end{bmatrix}, arc(6,7,24)=\begin{bmatrix}112 & 131\\ 84 & 103\\ 70 & 70\\ 98 & 145\end{bmatrix}$$

Arc [7 1]:

$$arc(7,1,1)=\begin{bmatrix}103 & 136\\ 75 & 122\\ 150 & 150\\ 89 & 94\end{bmatrix}, arc(7,1,2)=\begin{bmatrix}117 & 136\\ 75 & 122\\ 150 & 150\\ 89 & 108\end{bmatrix}, arc(7,1,3)=\begin{bmatrix}103 & 136\\ 89 & 94\\ 150 & 150\\ 75 & 122\end{bmatrix},$$

$$arc(7,1,4)=\begin{bmatrix}117 & 136\\ 89 & 108\\ 150 & 150\\ 75 & 122\end{bmatrix}, arc(7,1,5)=\begin{bmatrix}89 & 94\\ 75 & 122\\ 150 & 150\\ 103 & 136\end{bmatrix}, arc(7,1,6)=\begin{bmatrix}89 & 94\\ 103 & 136\\ 150 & 150\\ 75 & 122\end{bmatrix},$$

$$arc(7,1,7)=\begin{bmatrix}75 & 122\\ 103 & 136\\ 150 & 150\\ 89 & 94\end{bmatrix}, arc(7,1,8)=\begin{bmatrix}75 & 122\\ 117 & 136\\ 150 & 150\\ 89 & 108\end{bmatrix}, arc(7,1,9)=\begin{bmatrix}75 & 122\\ 89 & 94\\ 150 & 150\\ 103 & 136\end{bmatrix},$$

$$arc(7,1,10)=\begin{bmatrix}75 & 122\\ 89 & 108\\ 150 & 150\\ 117 & 136\end{bmatrix}, arc(7,1,11)=\begin{bmatrix}89 & 108\\ 75 & 122\\ 150 & 150\\ 117 & 136\end{bmatrix}, arc(7,1,12)=\begin{bmatrix}89 & 108\\ 117 & 136\\ 150 & 150\\ 75 & 122\end{bmatrix},$$

$$arc(7,1,13)=\begin{bmatrix}75 & 108\\ 103 & 136\\ 150 & 150\\ 89 & 122\end{bmatrix}, arc(7,1,14)=\begin{bmatrix}75 & 94\\ 89 & 136\\ 150 & 150\\ 103 & 122\end{bmatrix}, arc(7,1,15)=\begin{bmatrix}75 & 94\\ 103 & 122\\ 150 & 150\\ 89 & 136\end{bmatrix},$$

$$arc(7,1,16)=\begin{bmatrix}75 & 108\\ 89 & 122\\ 150 & 150\\ 103 & 136\end{bmatrix}, arc(7,1,17)=\begin{bmatrix}103 & 136\\ 75 & 108\\ 150 & 150\\ 89 & 122\end{bmatrix}, arc(7,1,18)=\begin{bmatrix}89 & 122\\ 75 & 108\\ 150 & 150\\ 103 & 136\end{bmatrix},$$

$$arc(7,1,19)=\begin{bmatrix}103 & 136\\ 89 & 122\\ 150 & 150\\ 75 & 108\end{bmatrix}, arc(7,1,20)=\begin{bmatrix}89 & 122\\ 103 & 136\\ 150 & 150\\ 75 & 108\end{bmatrix}, arc(7,1,21)=\begin{bmatrix}103 & 122\\ 75 & 94\\ 150 & 150\\ 89 & 136\end{bmatrix},$$

$$arc(7,1,22)=\begin{bmatrix}103 & 122\\ 89 & 136\\ 150 & 150\\ 75 & 94\end{bmatrix}, arc(7,1,23)=\begin{bmatrix}89 & 136\\ 75 & 94\\ 150 & 150\\ 103 & 122\end{bmatrix}, arc(7,1,24)=\begin{bmatrix}89 & 136\\ 103 & 122\\ 150 & 150\\ 75 & 94\end{bmatrix}$$

Arc [1 8]:

$$arc(1,8,1)=\begin{bmatrix}117 & 150\\ 89 & 136\\ 103 & 108\\ 75 & 75\end{bmatrix}, arc(1,8,2)=\begin{bmatrix}131 & 150\\ 89 & 136\\ 103 & 122\\ 75 & 75\end{bmatrix}, arc(1,8,3)=\begin{bmatrix}117 & 150\\ 103 & 108\\ 89 & 136\\ 75 & 75\end{bmatrix},$$

$$arc(1,8,4)=\begin{bmatrix}131 & 150\\ 103 & 122\\ 89 & 136\\ 75 & 75\end{bmatrix}, arc(1,8,5)=\begin{bmatrix}89 & 136\\ 117 & 150\\ 103 & 108\\ 75 & 75\end{bmatrix}, arc(1,8,6)=\begin{bmatrix}89 & 136\\ 131 & 150\\ 103 & 122\\ 75 & 75\end{bmatrix},$$

$$arc(1,8,7)=\begin{bmatrix}89 & 136\\ 103 & 108\\ 117 & 150\\ 75 & 75\end{bmatrix}, arc(1,8,8)=\begin{bmatrix}89 & 136\\ 103 & 122\\ 131 & 150\\ 75 & 75\end{bmatrix}, arc(1,8,9)=\begin{bmatrix}103 & 122\\ 131 & 150\\ 89 & 136\\ 75 & 75\end{bmatrix},$$

$$arc(1,8,10)=\begin{bmatrix}103 & 122\\ 89 & 136\\ 131 & 150\\ 75 & 75\end{bmatrix}, arc(1,8,11)=\begin{bmatrix}89 & 122\\ 117 & 150\\ 103 & 136\\ 75 & 75\end{bmatrix}, arc(1,8,12)=\begin{bmatrix}103 & 136\\ 117 & 150\\ 89 & 122\\ 75 & 75\end{bmatrix},$$

$$arc(1,8,13)=\begin{bmatrix}117 & 136\\ 103 & 150\\ 89 & 108\\ 75 & 75\end{bmatrix}, arc(1,8,14)=\begin{bmatrix}89 & 122\\ 103 & 136\\ 117 & 150\\ 75 & 75\end{bmatrix}, arc(1,8,15)=\begin{bmatrix}103 & 136\\ 89 & 122\\ 117 & 150\\ 75 & 75\end{bmatrix},$$

$$arc(1,8,16)=\begin{bmatrix}117 & 136\\ 89 & 108\\ 103 & 150\\ 75 & 75\end{bmatrix}, arc(1,8,17)=\begin{bmatrix}103 & 108\\ 117 & 150\\ 89 & 136\\ 75 & 75\end{bmatrix}, arc(1,8,18)=\begin{bmatrix}103 & 108\\ 89 & 136\\ 117 & 150\\ 75 & 75\end{bmatrix},$$

$$arc(1,8,19)=\begin{bmatrix}103 & 150\\ 117 & 136\\ 89 & 108\\ 75 & 75\end{bmatrix}, arc(1,8,20)=\begin{bmatrix}117 & 150\\ 103 & 136\\ 89 & 122\\ 75 & 75\end{bmatrix}, arc(1,8,21)=\begin{bmatrix}103 & 150\\ 89 & 108\\ 117 & 136\\ 75 & 75\end{bmatrix},$$

$$arc(1,8,22)=\begin{bmatrix}117 & 150\\ 89 & 122\\ 103 & 136\\ 75 & 75\end{bmatrix}, arc(1,8,23)=\begin{bmatrix}89 & 108\\ 103 & 150\\ 117 & 136\\ 75 & 75\end{bmatrix}, arc(1,8,24)=\begin{bmatrix}89 & 108\\ 117 & 136\\ 103 & 150\\ 75 & 75\end{bmatrix}$$

Arc [8 9]:

$$arc(8,9,1)=\begin{bmatrix}112 & 145\\ 84 & 131\\ 98 & 103\\ 70 & 70\end{bmatrix}, arc(8,9,2)=\begin{bmatrix}126 & 145\\ 84 & 131\\ 98 & 117\\ 70 & 70\end{bmatrix}, arc(8,9,3)=\begin{bmatrix}98 & 145\\ 84 & 103\\ 112 & 131\\ 70 & 70\end{bmatrix},$$

$$arc(8,9,4)=\begin{bmatrix}112 & 145\\ 84 & 117\\ 98 & 131\\ 70 & 70\end{bmatrix}, arc(8,9,5)=\begin{bmatrix}98 & 145\\ 112 & 131\\ 84 & 103\\ 70 & 70\end{bmatrix}, arc(8,9,6)=\begin{bmatrix}112 & 145\\ 98 & 131\\ 84 & 117\\ 70 & 70\end{bmatrix},$$

$$arc(8,9,7)=\begin{bmatrix}112 & 145\\ 98 & 103\\ 84 & 131\\ 70 & 70\end{bmatrix}, arc(8,9,8)=\begin{bmatrix}126 & 145\\ 98 & 117\\ 84 & 131\\ 70 & 70\end{bmatrix}, arc(8,9,9)=\begin{bmatrix}84 & 131\\ 112 & 145\\ 98 & 103\\ 70 & 70\end{bmatrix},$$

$$arc(8,9,10)=\begin{bmatrix}84 & 131\\ 126 & 145\\ 98 & 117\\ 70 & 70\end{bmatrix}, arc(8,9,11)=\begin{bmatrix}84 & 103\\ 98 & 145\\ 113 & 131\\ 70 & 70\end{bmatrix}, arc(8,9,12)=\begin{bmatrix}84 & 117\\ 112 & 145\\ 98 & 131\\ 70 & 70\end{bmatrix},$$

$$arc(8,9,13)=\begin{bmatrix}98 & 131\\ 112 & 145\\ 84 & 117\\ 70 & 70\end{bmatrix}, arc(8,9,14)=\begin{bmatrix}98 & 117\\ 126 & 145\\ 84 & 131\\ 70 & 70\end{bmatrix}, arc(8,9,15)=\begin{bmatrix}98 & 103\\ 112 & 145\\ 84 & 131\\ 70 & 70\end{bmatrix},$$

$$arc(8,9,16)=\begin{bmatrix}112 & 131\\ 98 & 145\\ 84 & 103\\ 70 & 70\end{bmatrix}, arc(8,9,17)=\begin{bmatrix}84 & 131\\ 98 & 103\\ 112 & 145\\ 70 & 70\end{bmatrix}, arc(8,9,18)=\begin{bmatrix}84 & 131\\ 98 & 117\\ 126 & 145\\ 70 & 70\end{bmatrix},$$

$$arc(8,9,19)=\begin{bmatrix}84 & 117\\ 98 & 131\\ 112 & 145\\ 70 & 70\end{bmatrix}, arc(8,9,20)=\begin{bmatrix}84 & 103\\ 112 & 131\\ 98 & 145\\ 70 & 70\end{bmatrix}, arc(8,9,21)=\begin{bmatrix}98 & 131\\ 84 & 117\\ 112 & 145\\ 70 & 70\end{bmatrix},$$

$$arc(8,9,22)=\begin{bmatrix}98 & 117\\ 84 & 131\\ 126 & 145\\ 70 & 70\end{bmatrix}, arc(8,9,23)=\begin{bmatrix}98 & 103\\ 84 & 131\\ 112 & 145\\ 70 & 70\end{bmatrix}, arc(8,9,24)=\begin{bmatrix}112 & 131\\ 84 & 103\\ 98 & 145\\ 70 & 70\end{bmatrix}$$

Arc [9 1]:

$$arc(9,1,1)=\begin{bmatrix}103 & 136\\ 75 & 122\\ 89 & 94\\ 150 & 150\end{bmatrix}, arc(9,1,3)=\begin{bmatrix}117 & 136\\ 75 & 122\\ 89 & 108\\ 150 & 150\end{bmatrix}, arc(9,1,4)=\begin{bmatrix}103 & 136\\ 89 & 94\\ 75 & 122\\ 150 & 150\end{bmatrix},$$

$$arc(9,1,4)=\begin{bmatrix}117 & 136\\ 89 & 108\\ 75 & 122\\ 150 & 150\end{bmatrix}, arc(9,1,5)=\begin{bmatrix}89 & 94\\ 75 & 122\\ 103 & 136\\ 150 & 150\end{bmatrix}, arc(9,1,6)=\begin{bmatrix}89 & 94\\ 103 & 136\\ 75 & 122\\ 150 & 150\end{bmatrix},$$

$$arc(9,1,7)=\begin{bmatrix}75 & 122\\ 103 & 136\\ 89 & 94\\ 150 & 150\end{bmatrix}, arc(9,1,8)=\begin{bmatrix}75 & 122\\ 117 & 136\\ 89 & 108\\ 150 & 150\end{bmatrix}, arc(9,1,9)=\begin{bmatrix}75 & 122\\ 89 & 94\\ 103 & 136\\ 150 & 150\end{bmatrix},$$

$$arc(9,1,10)=\begin{bmatrix}75 & 122\\ 89 & 108\\ 117 & 136\\ 150 & 150\end{bmatrix}, arc(9,1,11)=\begin{bmatrix}89 & 108\\ 75 & 122\\ 117 & 136\\ 150 & 150\end{bmatrix}, arc(9,1,12)=\begin{bmatrix}89 & 108\\ 117 & 136\\ 75 & 12\\ 150 & 150\end{bmatrix},$$

$$arc(9,1,13)=\begin{bmatrix}75 & 94\\ 89 & 136\\ 103 & 122\\ 150 & 150\end{bmatrix}, arc(9,1,14)=\begin{bmatrix}75 & 108\\ 103 & 136\\ 89 & 122\\ 150 & 150\end{bmatrix}, arc(9,1,15)=\begin{bmatrix}75 & 108\\ 89 & 122\\ 103 & 136\\ 150 & 150\end{bmatrix},$$

$$arc(9,1,16)=\begin{bmatrix}75 & 94\\ 103 & 122\\ 89 & 136\\ 150 & 150\end{bmatrix}, arc(9,1,17)=\begin{bmatrix}103 & 136\\ 75 & 108\\ 89 & 122\\ 150 & 150\end{bmatrix}, arc(9,1,18)=\begin{bmatrix}89 & 122\\ 75 & 108\\ 103 & 136\\ 150 & 150\end{bmatrix},$$

$$arc(9,1,19)=\begin{bmatrix}103 & 136\\ 89 & 122\\ 75 & 108\\ 150 & 150\end{bmatrix}, arc(9,1,20)=\begin{bmatrix}89 & 122\\ 103 & 136\\ 75 & 108\\ 150 & 150\end{bmatrix}, arc(9,1,21)=\begin{bmatrix}103 & 122\\ 75 & 94\\ 89 & 136\\ 150 & 150\end{bmatrix},$$

$$arc(9,1,22)=\begin{bmatrix}103 & 122\\ 89 & 136\\ 75 & 94\\ 150 & 150\end{bmatrix}, arc(9,1,23)=\begin{bmatrix}89 & 136\\ 75 & 94\\ 103 & 122\\ 150 & 150\end{bmatrix}, arc(9,1,24)=\begin{bmatrix}89 & 136\\ 103 & 122\\ 75 & 94\\ 150 & 150\end{bmatrix}$$

Comparing this example system to the example three product system with idle only in Section 5.3.2.1, makes it apparent that the output from the algorithm can quickly be overwhelmed by the sheer magnitude of data as the number of products increase. The three product system has two regions per arc while the four product system has 24 regions per arc. Currently the amount of CPU time for the algorithm to determine stable regions for the production system increases significantly with each increase in the number of products.

# 6 Product Sequencing Algorithm

## *6.1 Introduction*

Over the past few decades there has been much research dedicated to understanding and implementation of Just-In-Time (JIT) manufacturing (also known as Lean Manufacturing, Toyota Production System, or pull production) principles for various manufacturing systems. When examining a mixed-model assembly line, it is assumed that there are a variety of product models being assembled but all products will have similar characteristics to some degree. Such an assembly line allows the manufacturer to better meet the often diverse customer demand.

Proper product sequencing for a typical JIT system will yield a sequence in which a level load for each process in the line occurs as well as a constant rate of usage for each part on an assembly line. Monden [15] was one of the first to begin work in the area of level scheduling with the development of the Goal Chasing I and II methods. The goal of these methods is to minimize the variation on each process in the line and the variation on the speed of consuming each product at each step of the sequence. A level schedule (minimized usage rate variation) will be one in which each product is scheduled to be produced in a direct proportion to the level of demand for each product.

Miltenburg [25] continued the work of Monden [15] over many years and incorporated a measure of the level of product intermixing, called the usage rate variation. The deviation between actual usage rate and the desired usage rate of products was minimized to determine the best sequence. Miltenburg [38-40, 66, 81-84] proposed both algorithms and heuristics to solve this scheduling problem. This initial work assumed that setup times between differing products were negligible.

McMullen [54] continued the work of Miltenburg by adding a second objective of minimizing the number of setups. A weighting method was used to reach both objectives simultaneously, initially using a Tabu Search algorithm. In subsequent articles McMullen [55-59] proposed using Simulated Annealing, Genetic Algorithm, Kohonen Self-Organizing Map, an Ant Colony approach and an efficient frontier approach to solve the two objectives simultaneously.

Much of the existing research attempts to optimize the tradeoff between the number of setups to change from product to product and the smoothness of the sequence. The ideal sequence is one in which the usage rate variation and the number of setups are minimized for the sequence.

Pattern production has and continues to be researched and has been shown to be a very efficient method for production control of manufacturing systems with minimal setups between products. Signal Kanban production control has also been researched and implemented on the manufacturing floor as a method of controlling production when significant setups are present, although Toyota Georgetown no longer uses Signal Kanbans for production control. The previous work of Seidman and Holloway [7-8] analyzed a Signal Kanban with bounded demand fluctuation. The system proved to be stable for a given reorder point such that no backordered products ever occurred. A pattern production system was also examined and shown to be stable within a given range of system parameters. In [85], Holloway demonstrated that "fixed fill level" Signal Kanban policies are preferred to "fixed batch size" policies, because of potential problematic long-term cyclic behaviors in the "fixed batch size" policies. (Note that the systems in the dissertation are "fixed-fill," where production always fills a buffer to a specific level, instead of a "fixed batch size" refill.)

The purpose of this research is to lay the foundation for a sequencing algorithm that will function along a continuum between the two worlds of Signal Kanban production control and Pattern Production control. This general sequencing algorithm will replicate production sequences that would be present in either production control method depending on the parameters of the system. A signal kanban system can be mimicked by the sequencing algorithm by setting the buffer threshold levels at a mid-buffer level while a pattern production system can be mimicked by setting the buffer threshold levels to the maximum levels. This research also incorporates the common real-world problem of significant setups between products that may vary depending upon the production sequence. This method is not as computationally complex as the stability algorithm and can therefore be applied production systems with many products.

The product sequencing method is intended to be implemented for a JIT factory floor as an on-line production sequencing system. The algorithm is intended to be integrated into the production control system in order to receive real-time feedback on the production state and buffer state of each product.

## *6.2 Production System Model*

The production system being considered is one in which there are multiple products with potentially different production rates and usage rates and significant sequence dependent setups

between products. The production system is assumed to be a single stage system that can have idle time, see Figure 6.1. The system functions such that customer orders come into a "black box" of the sequencing algorithm as well as product information (current production conditions, buffer size and fullness levels, production and usage rates, setup costs, etc.). The algorithm processes the information and outputs a product to be produced next, which is passed to the production stage. The algorithm is intended to be updated and run after each product refill, where the sequence is based on real-time feedback of the system parameters. An alternative use is to run the algorithm to generate a short sequence of products at a given time interval, such as sequencing a day's worth of production determined each morning based on the current state of the production system.

The algorithm models a production system in which production occurs in batches, the batch size is the quantity of products required to fully replenish the buffer to a full level. When the product batch is completed, it is stored in Finished Goods Inventory (FGI) until a customer order is received and the required number of products are removed from FGI to meet the order. Buffer thresholds ($BF_{threshold,i}$) are defined for each product to signal the algorithm that the given product needs to be replenished. Only products at or below the buffer threshold are considered by the algorithm and if all products are above the buffer thresholds, the production system is idle, to replicate a lean system that only produces when customer demand is present.

The production system is assumed to behave in a deterministic manner such that the demand or usage rate ($UR_i$) can be represented as a constant value that does not change over time. The production rate ($PR_i$) is also assumed to be a constant value that does not change with time. The setup costs ($cost(i,j)$) used by the system are assumed to be a constant value, but may be dependent upon the previous product refilled. This means that the changeover cost to switch from product $i$ to product $j$ does not have to be equivalent to the cost to switch from product $i$ to product $k$.

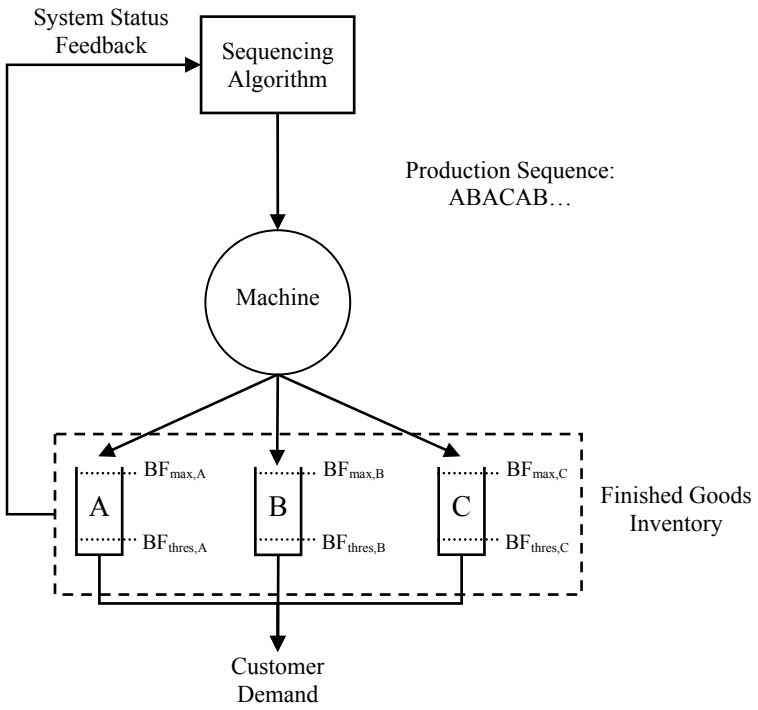


Figure 6.1: Production System Model

### *6.3 Time Normalized Method*

The method proposed for this research is one in which the system is time normalized in such a way that buffer levels and changeover costs are converted into units of real world time units. The first time conversion is to calculate the time to crash, $t_{crash,i}(t)$, which is the amount of time until the buffer of a given product is depleted. The time to crash is calculated by dividing the buffer level (number of products), $BF_i(t)$, by the demand or usage rate (number of products consumed per time unit), $UR_i$.

$$t_{crash,i}(t) = \frac{BF_i(t)}{UR_i} \tag{1}$$

As the time progresses into the future, the time to crash for all products will decrease an equivalent amount of time units. When product $i$ is refilled, the time to crash reaches the maximum value based on a full buffer which is given by the following equation:

$$t_{crash\_\max,i} = \frac{BF_{\max,i}}{UR_i} \tag{2}$$

From this equation it is obvious that the time to crash increases as the usage rate decreases or with an increase in the size of the buffer. A product with a low usage rate and small buffer could have an equivalent time to crash to a product that has a large buffer and high usage rate. Note that this term is not a function of time, but rather a constant term based on the parameters of product *i*.

The second time conversion is to determine the time required to refill the buffer of each product, $t_{refill,i}(t)$. The refill time is calculated by dividing the number of products missing from the buffer by the difference between the production rate *PR* and the usage rate of product *i*.

$$t_{refill,i}(t) = \frac{BF_{\max,i} - BF_i(t)}{PR_i - UR_i} \tag{3}$$

The maximum refill time occurs when the buffer is empty, which causes the equation to become the following:

$$t_{refill\_\max,i} = \frac{BF_{\max,i}}{PR_i - UR_i} \tag{4}$$

Similarly to the time to crash, the refill time increases with a decrease in production rate or usage rate, or an increase in the size of the buffer. Also two products with different production rates and buffer sizes could have equivalent times to refill. Also similar to the $t_{crash\_max,i}$, this term is not a function of time but rather a constant term based on the parameters of product *i*.

A plot of the time to crash versus time to refill can be constructed, assuming a constant usage rate and production rate, as shown in Figure 6.2. A straight line can be drawn between the maximum time to crash value $t_{crash\_max,i}$, and the maximum time to refill value $t_{refill\_max,i}$, for each product *i*. At any point in time the product buffer level will be located somewhere along (the solid dot) this line that connects a full buffer on the left end with an empty buffer on the right end. As seen in Figure 6.2, each line represents the characterization of a different product based upon the buffer size, production rate, and the usage rate. As products are consumed, the product state (the dot) will shift to the right and when the product is refilled, the state will be at the intercept with the vertical axis.

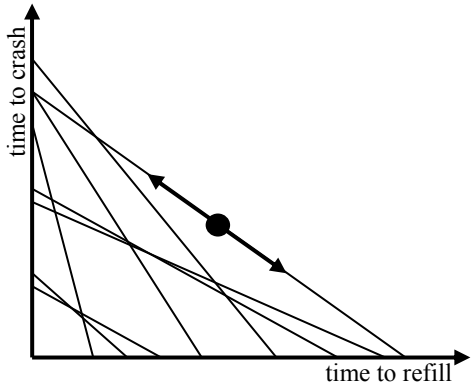


Figure 6.2: Plot of Refill Time versus Time to Crash

Note: Each line represents a different product.

The slope of the line for a given product can be calculated by using the two endpoints of the line to determine the "rise" ($t_{crash\_max,i}$) divided by the "run" ($t_{refill\_max,i}$) of the line. From Equations 2 and 4:

$$m_i = \frac{t_{crash\_max,i}}{t_{refill\_max,i}} = \frac{\left(\frac{BF_{\max,i}}{UR_i}\right) - 0}{0 - \left(\frac{BF_{\max,i}}{PR_i - UR_i}\right)} \tag{5}$$

Equation 5 will simplify to:

$$m_i = -\frac{PR_i - UR_i}{UR_i} \tag{6}$$

From the plot above, it is apparent that the maximum time to crash occurs when the time to refill is equal to zero, therefore if $t_{refill,i} = 0$, then

$$t_{crash,i}(0) = \frac{BF_{\max,i}}{UR_i} \tag{7}$$

By combining Equations 6 and 7 into the slope-intercept equation of a line ($y = mx + b$) where the time to crash is a function of the time to refill, also where

$$t_{crash,i}(t_{refill,i}) = y$$

and

$$b = t_{crash,i}(0) = -\frac{PR_i - UR_i}{UR_i} \times 0 + \frac{BF_{\max,i}}{UR_i}.$$

The equation for the $t_{crash,i}(t_{refill,i})$ of a given product $i$, as a function of time to refill, becomes the following equation:

$$t_{crash,i}(t_{refill,i}) = \frac{BF_{\max,i}}{UR_i} - \frac{PR_i - UR_i}{UR_i} t_{refill,i} \tag{8}$$

### 6.3.1 Lemma #1

*Lemma:*

For each product $i$, if the production system is not producing product $i$ over time period $[t, t + \Delta t]$, then

$$t_{crash,i}(t + \Delta t) = t_{crash,i} - \Delta t.$$

*Proof:*

This can be proven by considering some time $\Delta t$. This $\Delta t$ time is in time units and if $t_1 = t + \Delta t$ where $t$ is the initial time and the $t_1$ is the final time. The buffer of any product would be decreased according to the following equation:

$$BF_i(t_1) = BF_i(t) - UR_i \times \Delta t \tag{9}$$

Equation 9 can be reorganized to the following equation for all products:

$$\Delta t = \frac{BF_i(t) - BF_i(t_1)}{UR_i} \tag{10}$$

This can be further reorganized to the following:

$$\Delta t = \frac{BF_i(t)}{UR_i} - \frac{BF_i(t_1)}{UR_i} \tag{11}$$

Where $\frac{BF_i(t)}{UR_i}$ is the $t_{crash,i}$ at time $t$, which allows Equation 11 to be simplified to the following equation:

$$\Delta t = t_{crash,i}(t) - t_{crash,i}(t_1) \tag{12}$$

Which is equivalent to

$$t_{crash,i}(t_1) = t_{crash,i}(t) - \Delta t\ , \tag{13}$$

where $t_1 = t + \Delta t$. Therefore the time to crash for all products is decreased an equal amount $\Delta t$, as long as the product is not being refilled. If a product is refilled, the time to crash simply becomes equal to Equation 2, the maximum value of the time to crash.

□

### 6.3.2 Lemma #2

*Lemma:*

For each product $i$, if the production system is not producing product $i$ over time period $[t, t + \Delta t]$, then

$$t_{refill,i}(t + \Delta t) = t_{refill,i} + \frac{UR_i}{PR_i - UR_i}\Delta t\ .$$

*Proof:*

The proof begins by substituting $t + \Delta t$ into Equation 3.

$$t_{refill,i}(t + \Delta t) = \frac{BF_{\max,i} - BF(t - \Delta t)}{PR_i - UR_i}$$

$$= \frac{BF_{\max,i} - BF_i(t) + UR_i \times \Delta t}{PR_i - UR_i}$$

$$= \frac{BF_{\max,i} - BF_i(t)}{PR_i - UR_i} + \frac{UR_i \times \Delta t}{PR_i - UR_i}$$

Given that $t_{refill,i}(t) = \frac{BF_{\max,i} - BF_i(t)}{PR_i - UR_i}$, then

$$t_{refill,i}(t+\Delta t) = t_{refill,i}(t) + \frac{UR_i}{PR_i - UR_i}\Delta t \quad (14)$$

□

## *6.4 Quantifying Goodness of Products*

The sequencing algorithm evaluates all products that need replenishment and selects the next product to be replenished based upon a goodness calculation. The goodness calculation is computed for each product *i,* with an equation of five terms that are summed together and the product with the highest value is selected to be the next product in the production sequence. This section will provide an in depth examination of the goodness equation, first by defining key variables and then defining the terms of the equation, followed by a discussion of the behavior of the goodness equation.

### 6.4.1 Key Variables

- $BF\_threshold_i$: the buffer level at which product *i* must be at or below to signal the need for replenishment. This value is a percentage. When product *i* drops below the threshold, the algorithm will evaluate the product as possibly one of the next products in the production sequence.
- $BF_i(t)$: buffer level of product *i* at time *t*.
- $BF_{max,i}$: maximum buffer level of product *i*.
- $UR_i$: the usage rate of product *i*.
- $PR_i$: the production rate of product *i*.
- *prev_product*: the last product that was replenished by the production system at time *t.*
- *COST(prev_product,i)*: the cost of changing from *prev_product* to product *i*, in units of time.
- *la_time*: variable that stores the size of the lookahead window in units of time, which is how much time into the future the lookahead goodness will be calculated for all possible sequences.
- *MUR*: usage rate variation as calculated using equation developed by Miltenburg (Equation 1 in Chapter 2).
- *prod_thres*: variable that stores set of products that are at or below the $BF\_threshold_i$.
- *seq(k)*: set of all previous products for all previous stages, *k*, of production sequence.

- *prod_selection*: variable that stores which product is selected as the next product to be sequenced by the goodness calculations.
- Goodness Equation Weighting Factors: These factors are set by the user to manipulate the behavior of the goodness equation. The user defines the initial value of the factors in the range of [0,1] for each factor and sum of all factors is equal to one. The algorithm will normalize each factor based on the current state of the system. The corresponding normalized goodness equation term is equal to the initial weighting factor value. The normalization is implemented by the algorithm to prevent one term from dominating the goodness equation.
    - α: the weighting factor to control the "time to crash" term which is the time to reach an empty buffer.
    - β: the weighting factor to control the "time to refill" term which is the time required to completely refill the buffer from the current buffer level.
    - γ: the weighting factor to control the "time in queue" term which is the time that the product has been at or below the buffer threshold.
    - ε: the weighting factor to control the "changeover cost" term which is the time required to switch from the previous product to the current product.
    - η: the weighting factor to control the "usage rate variation" term as calculated using the equation developed by Miltenburg.

### 6.4.2 Terms of the Goodness Equation

The goodness equation contains five terms and each term has a weighting factor associated with the given term, the terms are: the time to crash, time to refill, time in queue, changeover cost, and usage rate variation.

The first term of the goodness equation is the "time to crash" term, which is a function of the usage rate and the buffer level of product *i* at time *t*, and has a weighting factor of α. This term is in units of time and it is a calculation of how much time until the buffer of product *i* is completely exhausted. The term is negative because selection of a product with a large time to crash is less desirable than selecting a product with a small time to crash.

$$"time\ to\ crash" = -\alpha \frac{BF_i(t)}{UR_i} \tag{15}$$

The "time to refill" term is the second term of the goodness equation and is a function of the production rate, the usage rate, maximum buffer size, and the buffer level of product *i* at time *t*. The time to refill term has a weighting factor of β. This term is in units of time and is a calculation of how much time is required to refill product *i* given the current buffer level at time *t*.

$$"time\ to\ refill" = \beta \frac{BF_{\max} - BF_i(t)}{PR_i - UR_i} \tag{16}$$

The "time in queue" is the third term in the goodness equation and is in units of time with a weighting factor of γ. This term is a function of the time when product *i* crosses the buffer threshold, $BF_{threshold,i}$, usage rate, maximum buffer size, current buffer level, and buffer threshold level .

$$"time\ in\ queue" = \begin{cases} 0 \text{ if } BF_i(t) > BF_{\max,i} \times BF_{threshold,i} \\ \gamma \times t_{threshold,i} \end{cases} \tag{17}$$

The time since a given product crosses the buffer threshold, $t_{threshold,i}$, is calculated using the usage rate, maximum buffer size, current buffer level, and buffer threshold level of the given product. Note that the $t_{threshold,i}$ calculation is only valid if the product is at or below the buffer threshold level, otherwise the $t_{threshold,i}$ is zero.

$$t_{threshold,i} = \frac{BF_{threshold,i} \times BF_{\max,i} - BF_i(t)}{UR_i} \tag{18}$$

The fourth term of the goodness equation is the "changeover cost" which has a weighting factor of ε. This term is independent of all system parameters except the changeover cost, which is the cost to change from the previous product *prev_product* to product *i*. This term requires the changeover cost to be defined in units of time. Note that this is a negative term because the selected product ideally will minimize changeover cost.

$$"changeover\ cost" = -\varepsilon COST(prev\_product, i) \tag{19}$$

The fifth and final term of the goodness equation is the "usage rate variation" term as calculated using the equation developed by Miltenburg (Equation 1 in Chapter 2), which uses a weighting factor of η. This is a dimensionless term and is the only term in the goodness equation that is not in time units. This term is a function of the sequence stage number, *k*, usage rate of product *i*, the cumulative usage rate for all products, and the previous products in the sequence, *seq* (if sequence is longer than one stage). This term is to be minimized in a production sequence, therefore the term is negative.

$$"usage\ rate\ variation" = f_{MUR}(UR_i, \sum_{i=1}^{I} UR_i, k, seq) = -\eta MUR \tag{20}$$

The different terms of the goodness equation change over time based on product usage and the previous product produced, assuming the production system is being examined at a time of choosing the next product to produce. The following relationships should be noted (see Figure 6.3 below for a graphical representation of these relationships):

1. "time to crash" term increases to a maximum value of zero with a slope of α (as follows from Lemma #1). The minimum value of the time to crash occurs with a full buffer and is calculated $-\alpha \frac{BF_{\max,i}}{UR_i}$.

2. "time to refill" term increases with time with a slope of $\beta \frac{UR_i}{PR_i - UR_i}$ (as follows from Lemma #2). The maximum value of the time to refill term occurs when the buffer is empty and is calculated $\beta \frac{BF_{\max,i}}{PR_i - UR_i}$.

3. "time in queue" term increases with time with a slope of γ if the product is currently in the queue, meaning that the product is below $BF_{threshold,i}$. The maximum value of the time in queue term occurs with an empty buffer and is calculated $\gamma \frac{BF_{threshold,i} \times BF_{\max,i}}{UR_i}$.

4. "changeover cost" term does not change with respect to time, the term is dependent only upon the previously produced product and product *i* that is being considered by the goodness equation.

5. "usage rate variation" term does not change with respect to time, the term is dependent upon the cumulative usage rate for all products, the previously produced product sequence, and product *i* that is being considered by the goodness equation.

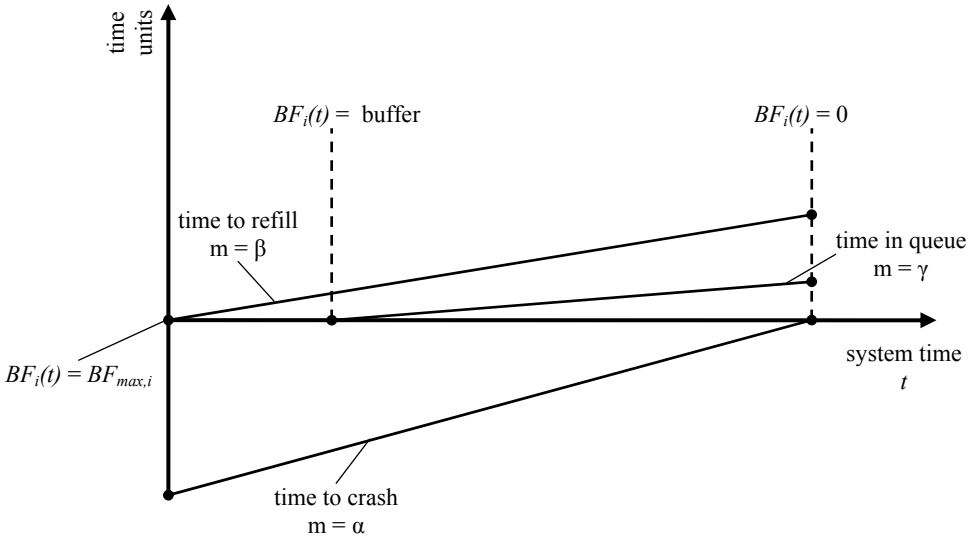


Figure 6.3: Plot of Example Goodness Equation Terms
Note: The terms are plotted over time, starting with a full buffer and ending with an empty buffer.

### 6.4.3 Weighting Factors of the Goodness Equation

Each term of the goodness equation has a weighting factor that allows the user to control the performance of the algorithm. The goodness equation is a function of five weighting variables and each term of the equation can be manipulated by a change in these weighting variables. The ability to weight the importance of each term of the goodness equation allows the algorithm to synthesize various known and proven lean production control systems. For instance, placing all the priority on the "time in queue" term of the equation will replicate a Signal Kanban system, where the sequencing will be completely determined by the order of products falling below their thresholds.

The weighting factors α, β, γ, ε, and η will change each respective term of the goodness equation. The slope of each term will be increased or decreased with an increase or decrease in the respective weighting factor. All weighting factors will maintain the following constraints at all times.

$$0 \leq \alpha_{ini} \leq 1$$

$$0 \le \beta_{ini} \le 1$$

$$0 \le \gamma_{ini} \le 1$$

$$0 \le \varepsilon_{ini} \le 1$$

$$0 \le \eta_{ini} \le 1$$

$$\alpha_{ini} + \beta_{ini} + \gamma_{ini} + \varepsilon_{ini} + \eta_{ini} = 1$$

Under these conditions, a maximum slope of each term occurs when the weighting factor is equal to one. This means that the maximum slope of the "time to crash" is one, as well as the "time in queue" term. The maximum slope of the "time to refill" term is $\frac{UR_i}{PR_i - UR_i}$, when β equals one. The changeover cost and usage rate variation terms are piecewise horizontal lines with a maximum value of ε or η, respectively.

The importance of the weighting factors is that it allows the user to determine which system parameters can potentially choose the next product to be refilled. For example consider two products with equal usage rates, buffer sizes, and buffer thresholds but the production rate of product B is half of the rate of product A. If β is very small in comparison to α and/or γ, then the two products would appear as equivalent products to the algorithm. This is because both the "time to crash" and "time in queue" terms are not functions of the production rate.

When α is very large (~1) when compared to β and γ, two products will be considered equivalent if the ratios of buffer size to usage rate is equal for both products, assuming both were refilled at exactly the same time ($t_{refill,A} = t_{refill,B}$). When β is very large compared to α or γ, two products will be considered as equivalent products if the ratios of usage rate to production rate are equal, again assuming both were refilled at exactly the same time ($t_{refill,A} = t_{refill,B}$). Similarly γ heavy products are considered equivalent products if the buffer threshold is crossed at the same time for both products. This means the products could have different usage and production rates, buffer sizes, and buffer thresholds.

Note that the initial weighting factor ($X_{ini}$) is within a range of zero to one, but each factor is normalized ($X$) prior to being used in the goodness equation. This allows the algorithm to more accurately consider each term of the equation because the weighting factors are normalized such that the maximum value of each term is no larger than one for all products.

Consider the time to crash and the changeover cost terms for a product that has a large buffer and slow usage rate. This implies that the time to crash term is very large, perhaps a value in the hundreds of hours and assume that the changeover term is less than an hour. Without a normalized weighting factor, the time to crash term will dominate the changeover term and the changeover cost will not affect the sequence until the time to crash is less than an hour. Therefore each term of the goodness equation is normalized to a value of one to increase the effectiveness of each factor.

The normalized coefficient is determined first by calculating the maximum value of the term that the coefficient will be applied to (i.e. time to crash, time to refill, time in queue, changeover cost, or Miltenburg's usage rate variation). The maximum value of each goodness equation term is calculated for the products contained in the *prod_thres*, which is the set of products that are below the buffer threshold, $BF_{threwshold,i}$. The initial value of the weighting factor is then divided by the corresponding normalizing coefficient variable and the new normalized weighting factor value is used for the goodness calculations.

$$\alpha = \frac{\alpha_{ini}}{\max(time\_to\_crash_{i \in prod\_thres})}$$

$$\beta = \frac{\beta_{ini}}{\max(time\_to\_refill_{i \in prod\_thres})}$$

$$\gamma = \frac{\gamma_{ini}}{\max(time\_in\_queue_{i \in prod\_thres})}$$

$$\varepsilon = \frac{\varepsilon_{ini}}{\max(\text{cost})}$$

$$\eta = \frac{\eta_{ini}}{MUR_{prev}}$$

## *6.5 Method of Product Selection*

The goodness equation quantifies all products at or below the buffer threshold either for the current state of the system or for a future product sequence when lookahead time is considered. The products are ranked from highest to lowest value of goodness equation and the highest valued product is selected as the next product or the first product of the highest valued future sequence is selected as the next product. Note that the goodness value is relative based on the

current state of the products and system and is not an absolute calculation. The goodness equation is shown below, note that the time to crash, changeover cost, and usage rate variation terms are negative. The negative sign is used because products that have high values of these terms is less desirable to be the next product than a product with lower values of time to crash, changeover cost, and usage rate variation.

$goodness_i(t) =$

$$-\alpha\frac{BF_i}{UR_i}+\beta\frac{BF_{\max,i}-BF_i}{PR_i-UR_i}+\gamma(t-t_{threshold,i})-\varepsilon COST(prev\_product,i)-\eta MUR_{current} \quad (21)$$

The following statements characterize the two methods used by the algorithm, first without lookahead and second with lookahead, to select a product to be sequenced. The state of the production system is described as a function of time by the following:

$$S(t)=(UR_i,BF_i,BF_{\max,i},BF_{threshold,i},PR_i,UR_i,COST,seq) \quad (22)$$

### 6.5.1 Current State Decision Statement

Given a decision time $t$ and current state of the production system $S(t)$, select the product $i$ that is at or below $BF_{threshold,i}$ that has the maximum goodness equation value when calculated with the normalized weighting factors.

### 6.5.2 Lookahead State Decision Statement

Given a decision time $t_o$, a lookahead time *la_time*, and current state of the production system $S(t_o)$, a product sequence tree (see Figure 6.4) is generated of all possible production sequences that do not experience empty buffers, $SEQ(S(t_o),la\_time)$, from the current time $t_o$ to the first decision time greater than $t_o$ + *la_time*. Where each production sequence contained in $SEQ(S(t_o),la\_time)$ begins with a product that is below $BF_{threshold,i}$ at time $t_o$. All sequence tree branches represent a product being selected to be produced at decision time $t_{future}$ where the system is characterized by $S(t_{future})$.

Product $i$ is selected to be the next product to be produced using lookahead goodness criteria if product $i$ is the first product of *seq_star*, where *seq_star* is the sequence in $SEQ(S(t_o),la\_time)$ with the highest average goodness value over the entire sequence branch. An example in Section 6.6 is provided to clarify the method for selecting a product with lookahead.

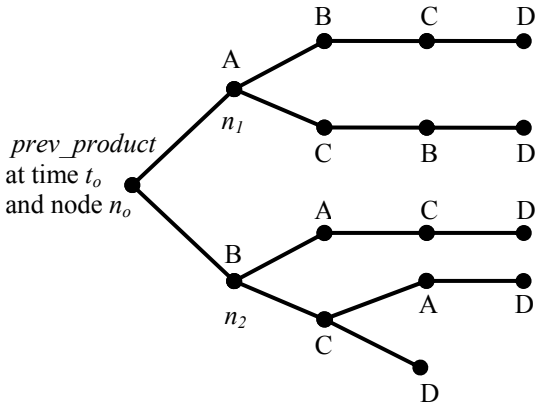


Figure 6.4: Sequencing Tree

Note that Figure 6.4 is a conventional sequencing tree for system of four products and it shows all possible branches that originate from the current state of the system at time $t_o$. At time $t_o$ there are only branches for choosing product *A* or *B*, as those are the only products with inventory levels below the threshold at time $t_o$ in this example.

#### 6.5.2.1 Alternative Lookahead State Decision Method

The averaged goodness values over the steps of sequence branch could potentially obscure a low goodness value in an otherwise high average sequence. An alternative lookahead selection method is proposed in which the sequence with the maximum of all minimum goodness values for all sequences over the lookahead time is selected as the"best" sequence. The first product of the "best"sequence is selected as the next product in the production sequence.

Consider two sequences where each sequence has five stages. Sequence A has goodness values of 10 for four stages and a goodness value of one for one stage, with an average goodness value of 8.2 for the sequence. Sequence B has a goodness value of 8 for all five stages and average value of 8 for the entire sequence. Selecting a sequence based on the average goodness will select sequence A. Consider selecting a sequence with the maximum of the minimum goodness value of all the sequences as an alternative selection method. This alternative method would select sequence B and avoid selecting a sequence with a low value.

## *6.6 Example of Goodness Equation with Lookahead*

The use of lookahead time does not change the goodness equation behavior or use of each term/weighting factor. The use of lookahead requires all potential sequences into the future to be

considered and the goodness at each stage is calculated and averaged, although the final average goodness over the sequence is considered when selecting the next product to be sequenced at time $t_o$. Each potential sequence is generated by updating the state of the system $S(t_{future})$ and considering all products that are at or below the buffer threshold level to generate all possible sequence branches until the lookahead time is reached. The goodness is calculated at each stage of the sequence but the average goodness for the entire branch is used to rank the product sequences. The "best" product that is selected as the next product is the first product of the sequence with the highest average goodness value at the lookahead time.

Consider the example system in Figure 6.5, which is a production system that consists of four products, A, B, C, and D. Note that the horizontal length of each product box represents the time to refill the respective product as time progresses from left to right. The goodness is calculated initially at time $t_o$ for products A and B (at each diamond) because these are the only products at or below the buffer threshold. The product with the highest goodness will be selected as the next product to be sequenced, if lookahead is not being considered. Note that the lookahead window spans some time into the future, denoted by the dotted box. The other products C and D in the lookahead window cross the buffer threshold at a future time, at which point the products are then considered in the future goodness calculations (at each dot). When lookahead is considered, the goodness is calculated at each stage of the sequence but all the goodness values are averaged in order to rank the product sequences. The sequence with the highest goodness is selected, such as A-B-C-D, where product A will be selected as the next product to be sequenced.

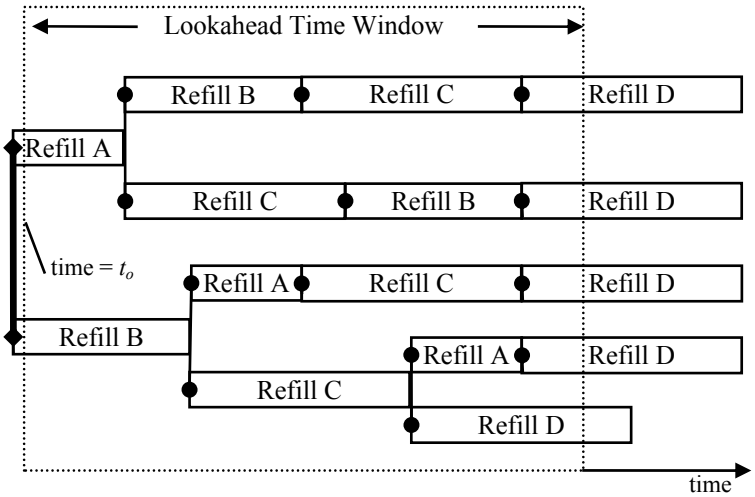


Figure 6.5: Sequencing Choices Over Time

Consider another hypothetical example to be evaluated using lookahead in which three products A, B, and C are below the buffer threshold and product *D* was just refilled. The goodness equation returns a goodness value of 9 for product *A*, 9 for product *B*, and 9 for product *C* when each respective product is the first product to be refilled in the lookahead sequence. The lookahead time value has not been reached yet, so the lookahead equation is used again to consider the next product in the lookahead sequence. After refilling product A in the *D-A* branch in the product sequence tree shown below, product *B* or *C* can be refilled. The goodness values are calculated for these products and a value of -1 is found for product *B* and a value of 1 for product *C*. In the *D-B* branch the goodness values are calculated to be 6 for product *A* and a value of 4 for product *C*. In the *D-C* branch the goodness values are calculated to be -1 for product *A* and a value of -1 for product *B*. Given the system parameters, the lookahead time is reached after refilling the second product, so the sequences are sorted and ranked to maximize the final goodness values. The lookahead sequence *D-B-A* is selected as the best sequence with an average goodness value of 7.5, therefore product *B* is the next product to be refilled. Note that the alternative lookahead selection method of maximizing the minmum value would make the same selection.

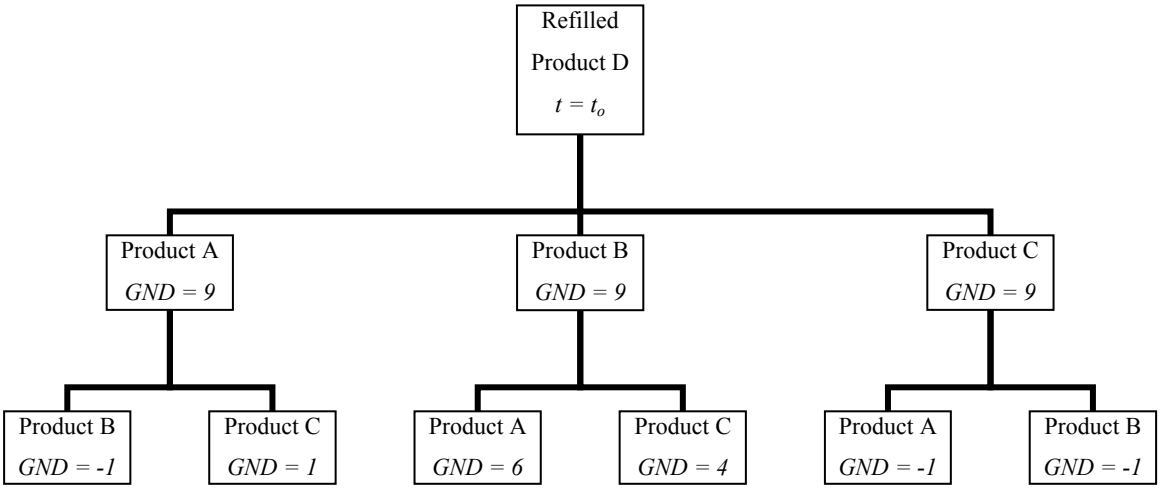


Figure 6.6: Product Sequence Tree

The advantage of lookahead is that the sequencing algorithm can look into the future to avoid dead-end branches in the product sequence tree. In this example, consider what would happen if product *B* has a very high usage rate and small buffer. After the first round of goodness calculations, all products are equal. The differences between the products is not apparent until the second round of goodness calculations which returns very low goodness values for the *D-A*

and *D-C* branches. This implies that the buffer of product *B* is depleted and the system will crash if these branches are chosen.

## *6.7 Sequencing Examples*

This section will provide examples and simulated results of the goodness sequencing method for various hypothetical production systems and examine the effects of varying different parameters. An explanation of the implementation and source code is provided in Appendix II.

### 6.7.1 Three Product Production System

Consider a system with three products with the node/arc network show below and the following parameters for each product: setup time = 5 time units, production rate = 10 products/time unit, usage rate = 1 product/time unit, lower threshold = 25 products, full buffer level = 100 products, initial buffer level = 100 products. In this network, all product sequences may pass through the idle node prior to being replenished or a product can wait in a queue to enter setup directly after replenishment of the previous product. All weighting factors are set equal to 0.2.

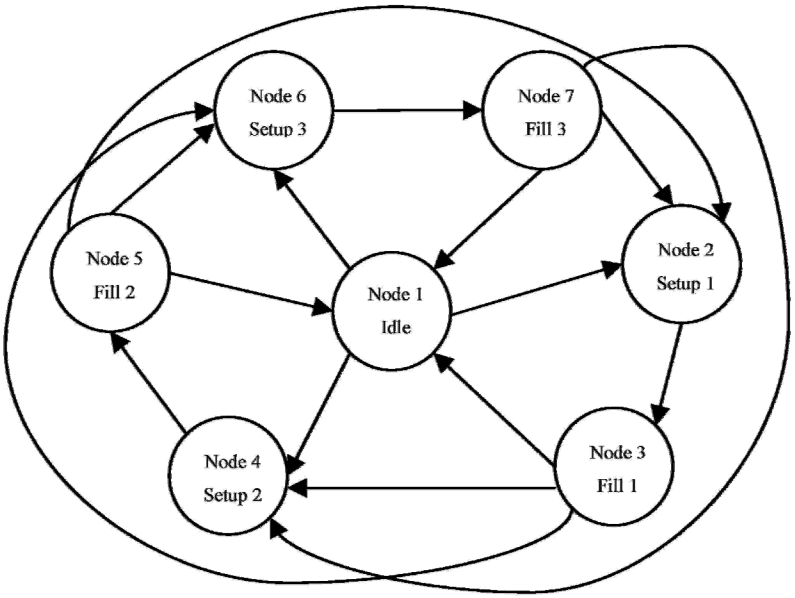


Figure 6.7: Network Map of Three-Product System – With and Without Idle

This is a very simplistic production system but will highlight the methods of the sequencing algorithm. Given that all products are equal there are only two possible repeated sequences: 1–2–3 or 1–3–2. A plot of the first 50 steps of the sequence is below plotted as percentage of buffer fullness versus time.

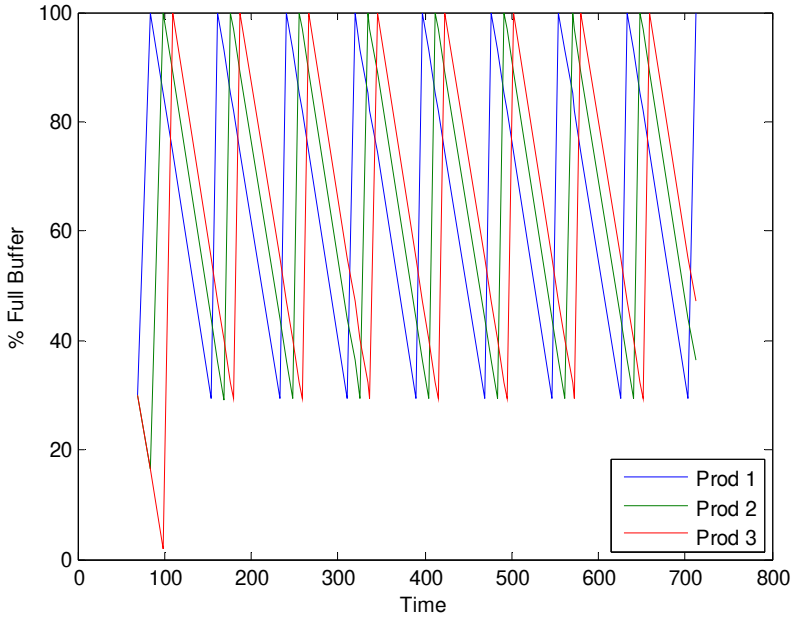


Figure 6.8: Plot of Three Product Sequence

From the startup time until 60 units of time, the system is idle because all products are above the buffer threshold level of 40 products. The first 125 to 150 units of time are required to deplete the initial buffer supply and refill the buffers to reach a non-transient sequence of products 1– 2– 3. The output of the algorithm for this system is unaffected by the use of the lookahead function and weighting factors for both goodness equations.

Consider doubling the usage rate of product one to two products consumed per time unit and all other parameters remain the same. This system crashes within the first 85 units of time after startup when either product one or three is depleted before both can be refilled. A plot of the sequence is below in terms of percentage of buffer fullness versus time.

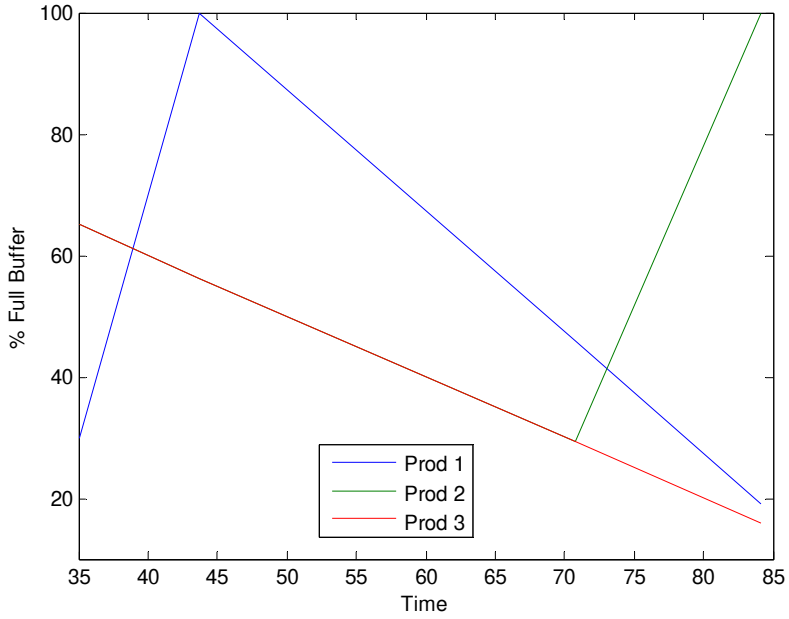


Figure 6.9: Plot of Three Product Sequence – Product 1 Increased Usage Rate

This system crash can be fixed by changing the initial buffer levels to disperse the triggering time of the products or by using the lookahead feature of the algorithm. A plot of the output from the algorithm is below in which the initial buffers are set to 50 products for all products.

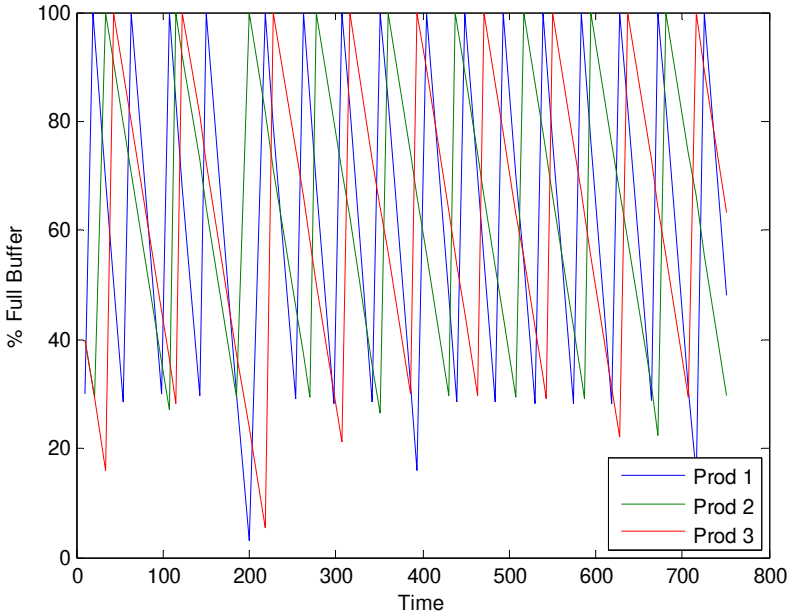


Figure 6.10: Plot of Three Product Sequence – Decreased Initial Buffer Levels

Notice in the plot above that the sequence of products appears to be random and chaotic in nature. Consider the output from the sequencing algorithm if the buffer threshold for product number one is increased to 40% from 30% and the initial buffers are all full. The plot of the sequence output for this set of parameters is below. The plot makes it apparent that the system exhibits a repeating pattern in the production sequence of 1– 2– 3–1 after the initial transient startup.

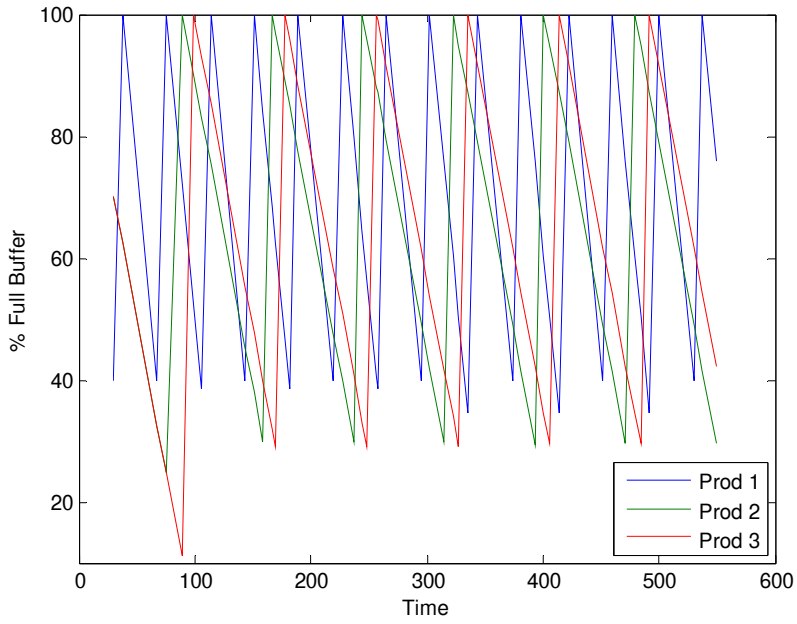


Figure 6.11: Three Product Sequence – Product 1 Higher Usage Rate and Buffer Threshold

### 6.7.2 Eight Product Production System

As the number of products in a production system increases, evaluation of production sequences can easily become unmanageable when evaluating with hand calculations. This sequencing algorithm provides feasible results for large production systems in a more expedient manner than the method in Chapter 4. Consider an eight product production system with the following parameters for all products: equal goodness weighting factors, buffer threshold level = 95% of maximum buffer levels, initial buffer level = 90% of maximum buffer levels. The production rates, usage rates, and maximum buffer levels vary for each product according to the following matrices:

$$PR(i) = \begin{bmatrix} 23 & 90 & 90 & 23 & 45 & 45 & 12 & 23 \end{bmatrix}$$

$$UR(i) = \begin{bmatrix} 2 & 2 & 2 & .25 & .25 & .25 & 1 & 1 \end{bmatrix}$$

$$BF_{\max}(i) = \begin{bmatrix} 200 & 100 & 200 & 200 & 100 & 200 & 200 & 100 \end{bmatrix}$$

$$COST(prev\_prod, i) = \begin{bmatrix} 3 & 3 & 6 & 3 & 6 & 3 & 6 & 3 \\ 3 & 3 & 3 & 6 & 3 & 3 & 3 & 3 \\ 3 & 6 & 3 & 3 & 6 & 3 & 3 & 3 \\ 3 & 3 & 6 & 3 & 3 & 6 & 6 & 3 \\ 3 & 6 & 3 & 3 & 3 & 6 & 6 & 3 \\ 3 & 3 & 6 & 3 & 6 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

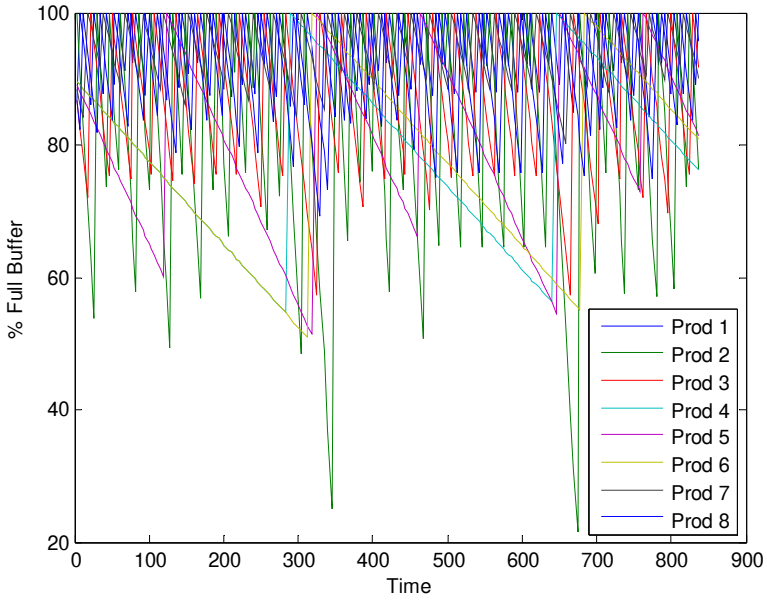


Figure 6.12: Plot of Eight Product Sequence without Lookahead

A sequence for this system can also be found by the algorithm using the lookahead function with a lookahead of 25 time units. The output sequence is plotted below for the eight product system using lookahead.

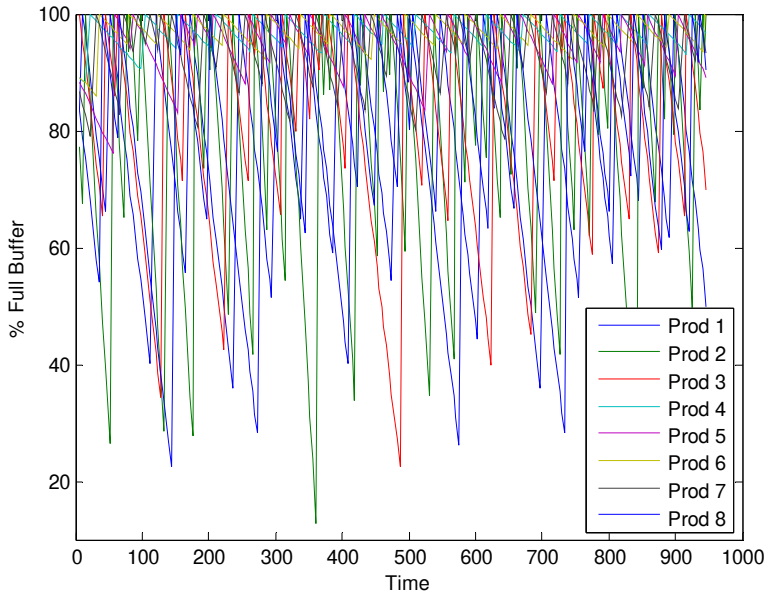


Figure 6.13: Plot of Eight Product Sequence with Lookahead

### 6.7.3 Weighting Parameters

The weighting parameters for the goodness equation and lookahead goodness equation allow the user to influence the output sequence from the algorithm. The weighting parameters increase or decrease the effect of the time to crash, time to refill, time in queue, changeover cost, or usage rate variation on the goodness calculation at each step of the sequence.

In order to generate the figures in this section, one weighting factor is considered to be a dominant factor in the goodness equation as the value varies from 0 to 1. The dominant factor is considered to be variable *a* in the following equation.

$$a + b = 1$$

The other weighting factors change as the dominate factor changes ($b = 1 - a$), where the sum of all weighting factors is always equal to one. There are 40 equally spaced points as *a* varies from 0 to 1, and a sequence of 75 stages is simulated at each data point. As an example of how the weighting factors vary in the simulations in this section, consider α as the dominant factor and all other factors equally weighted, (i.e. $\beta = \gamma = \varepsilon = \eta = \frac{b}{4}$). Initially, when $a$ = 0 and $b$ = 1, the weighting factors are $\alpha = 0$ and $\beta = \gamma = \varepsilon = \eta = \frac{1}{4}$. The last data point where $a$ = 1 and $b$ = 0, the weighting factors are $\alpha = 1$ and $\beta = \gamma = \varepsilon = \eta = 0$.

An eight product production system is considered for the test cases in this section, with the only variant (other than the weighting factors) being whether or not lookahead is present in the simulation. The production system has the following parameters for all products: buffer threshold level = 99.5% of maximum buffer levels, initial buffer level = 100% of maximum buffer levels. The production rates, usage rates, and maximum buffer levels for the products are defined according to the following matrices:

$$PR(i) = \begin{bmatrix} 23 & 90 & 90 & 23 & 45 & 45 & 12 & 23 \end{bmatrix}$$

$$UR(i) = \begin{bmatrix} 2 & 2 & 2 & .25 & .25 & .25 & 1 & 1 \end{bmatrix}$$

$$BF_{\max}(i) = \begin{bmatrix} 200 & 100 & 200 & 200 & 100 & 200 & 200 & 100 \end{bmatrix}$$

$$COST(prev\_prod, i) = \begin{bmatrix} 3 & 3 & 6 & 3 & 6 & 3 & 6 & 3 \\ 3 & 3 & 3 & 6 & 3 & 3 & 3 & 3 \\ 3 & 6 & 3 & 3 & 6 & 3 & 3 & 3 \\ 3 & 3 & 6 & 3 & 3 & 6 & 6 & 3 \\ 3 & 6 & 3 & 3 & 3 & 6 & 6 & 3 \\ 3 & 3 & 6 & 3 & 6 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{bmatrix}$$

#### 6.7.3.1 Configuration #1-A – Time to Crash Without Lookahead

This configuration evaluates the output from the algorithm in terms of percentage of full buffer when α is the dominant weighting factor for the time to crash term. The other factors β, γ, ε, and η are equal to one another. The first test case for configuration number one is without the lookahead function. The weighting factors are defined as functions of $a$ and $b$ with $\alpha = a$ and. The maximum average buffer level is 90.4% and the minimum is 81.8%.

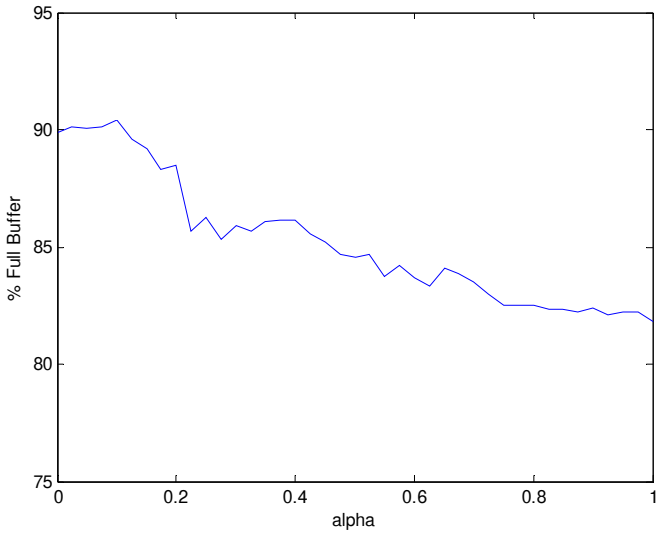


Figure 6.14: Configuration #1A Without Lookahead for Example State

#### 6.7.3.2 Configuration #1-B – Time to Crash With Lookahead

This configuration maintains the same values of weighting factors for the goodness equation as in the configuration #1-A test case. Lookahead time of 15 units is considered in this test case. The maximum average buffer level is 87.8% and the minimum is 84.2%.

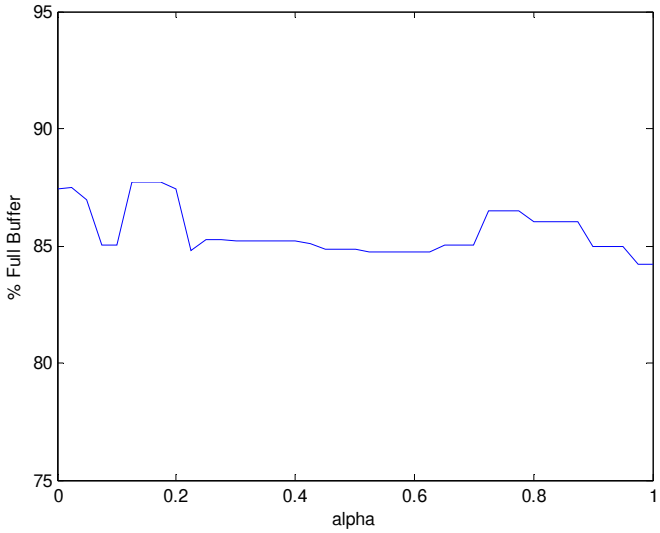


Figure 6.15: Configuration #1B With Lookahead for Example State

#### 6.7.3.3 Configuration #2-A – Time to Refill Without Lookahead

This configuration evaluates the output from the algorithm in terms of percentage of full buffer when β is the dominant weighting factor for the time to refill term. The other factors α, γ, ε, and η are equal to one another. The first test case for configuration number two is without the

lookahead function. The weighting factors are defined as functions of $a$ and $b$ with $\beta = a$ and $\alpha = \gamma = \varepsilon = \eta = \frac{b}{4}$. The maximum average buffer level is 88.7% and the minimum is 86.0%.

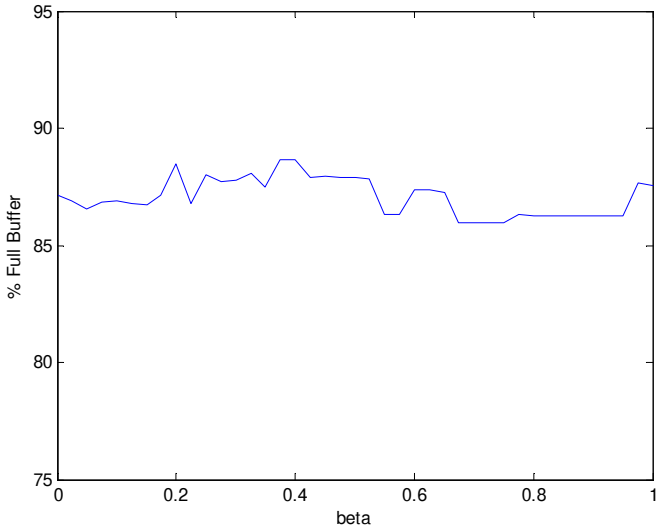


Figure 6.16: Configuration #2A Without Lookahead for Example State

6.7.3.4 Configuration #2-B – Time to Refill With Lookahead

This configuration maintains the same values of weighting factors for the goodness equation as in the configuration #2A test case. Lookahead time of 15 units is considered in this test case. The maximum average buffer level is 88.6% and the minimum is 83.9%.

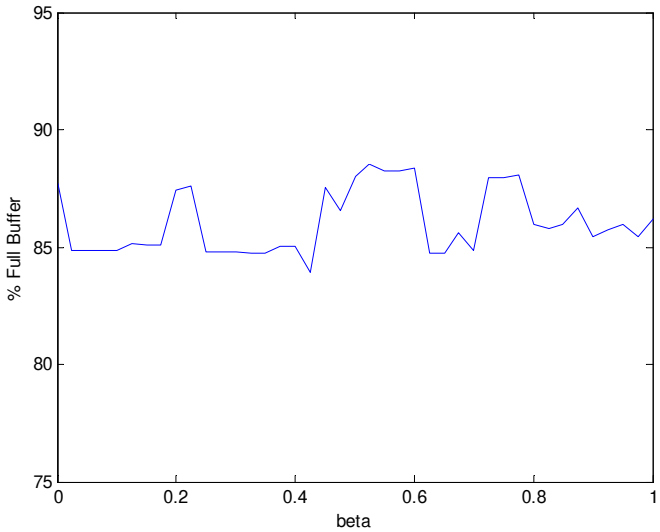


Figure 6.17: Configuration #2B With Lookahead for Example State

#### 6.7.3.5 Configuration #3-A – Time in Queue Without Lookahead

This configuration evaluates the output from the algorithm in terms of percentage of full buffer when γ is the dominant weighting factor for the time in queue term. The other factors α, β, ε, and η are equal to one another. The first test case for configuration number three is without the lookahead function. The weighting factors are defined as functions of $a$ and $b$ with $\gamma = a$ and $\alpha = \beta = \varepsilon = \eta = \frac{b}{4}$. The maximum average buffer level is 90.4% and the minimum is 84.8%.

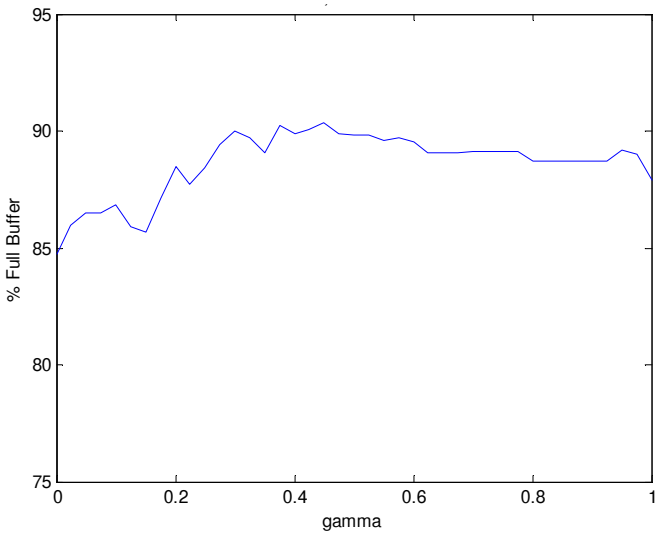


Figure 6.18: Configuration #3A Without Lookahead for Example State

#### 6.7.3.6 Configuration #3-B – Time in Queue With Lookahead

This configuration maintains the same values of weighting factors for the goodness equation as in the configuration #3A test case. Lookahead time of 15 units is considered in this test case. The maximum average buffer level is 87.3% and the minimum is 84.9%.

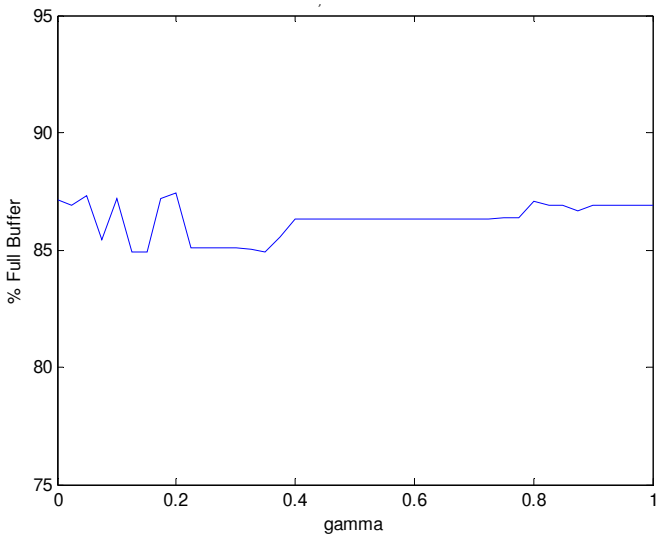


Figure 6.19: Configuration #3B With Lookahead for Example State

#### 6.7.3.7 Configuration #4-A – Changeover Cost Without Lookahead

This configuration evaluates the output from the algorithm in terms of percentage of full buffer when ε is the dominant weighting factor for the changeover cost term. The other factors α, β, γ, and η are equal to one another. The first test case for configuration number four is without the lookahead function. The weighting factors are defined as functions of $a$ and $b$ with $\varepsilon = a$ and $\alpha = \beta = \gamma = \eta = \frac{b}{4}$. The maximum average buffer level is 88.5% and the minimum is 83.9%.

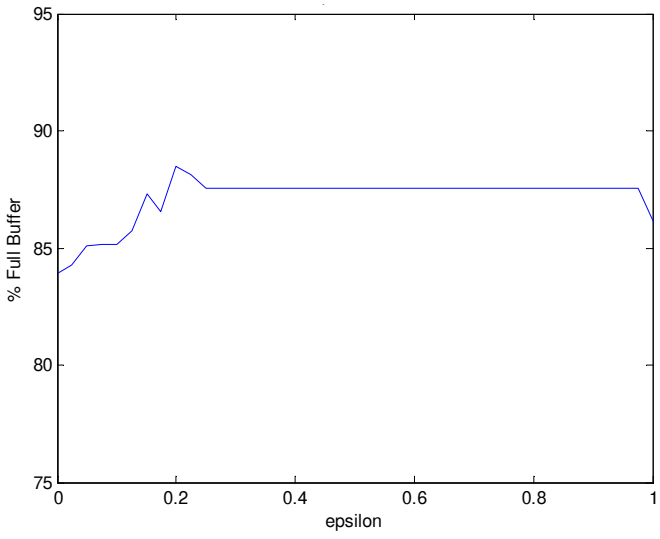


Figure 6.20: Configuration #4A Without Lookahead for Example State

#### 6.7.3.8 Configuration #4-B – Changeover Cost With Lookahead

This configuration maintains the same values of weighting factors for the goodness equation as in the configuration #4A test case. Lookahead time of 15 units is considered in this test case. The maximum average buffer level is 89.4% and the minimum is 84.2%.

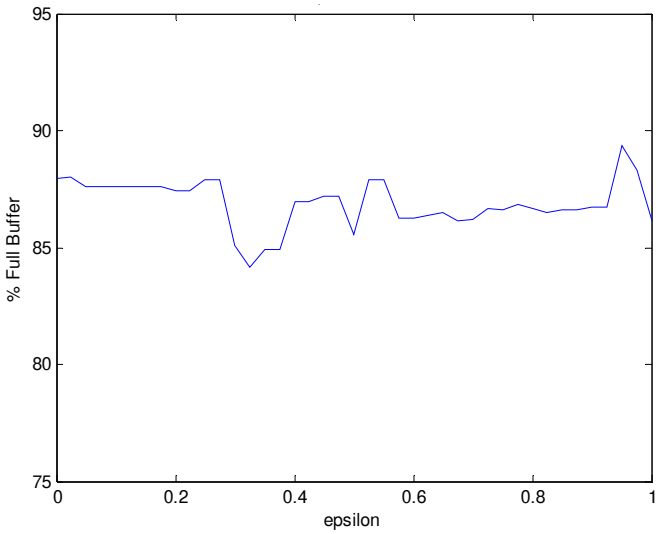


Figure 6.21: Configuration #4B With Lookahead for Example State

#### 6.7.3.9 Configuration #5-A – Usage Rate Variation Without Lookahead

This configuration evaluates the output from the algorithm in terms of percentage of full buffer when η is the dominant weighting factor for the usage rate variation term. The other factors α, β, γ, and ε are equal to one another. The first test case for configuration number five is without the lookahead function. The weighting factors are defined as functions of $a$ and $b$ with $\eta = a$ and $\alpha = \beta = \gamma = \varepsilon = \frac{b}{4}$. The maximum average buffer level is 88.5% and the minimum is 85.3%.

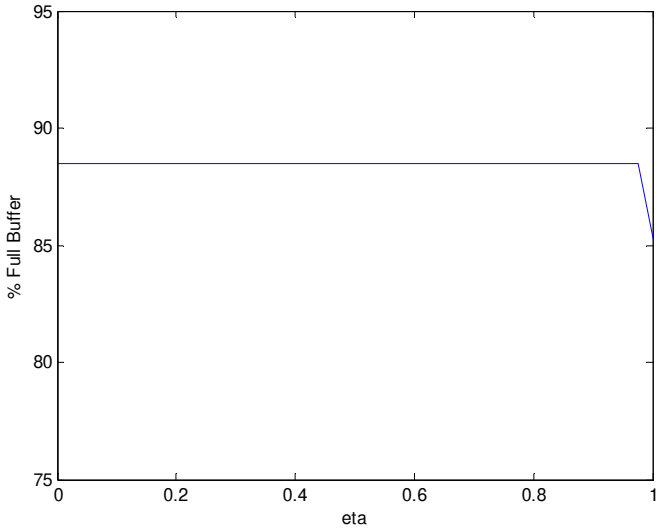


Figure 6.22: Configuration #5A Without Lookahead for Example State

#### 6.7.3.10 Configuration #5-B – Usage Rate Variation With Lookahead

This configuration maintains the same values of weighting factors for the goodness equation as in the configuration #5A test case. Lookahead time of 15 units is considered in this test case. The maximum average buffer level is 88.9% and the minimum is 87.4%.

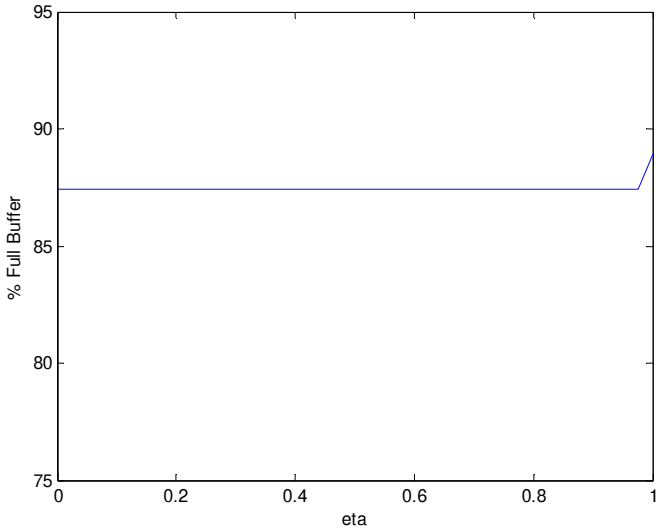


Figure 6.23: Configuration #5B With Lookahead for Example State

#### 6.7.3.11 Discussion of Weighting Factors Results

All test cases begin with the dominant weighting factor at the minimum value of zero when *a* equals zero. The non-dominant factors all have a beginning value of 0.25 so that the non-

dominant factors are all equal and sum to one. As the dominant factor increases from zero to one, the non-dominant factors decrease from a maximum value of 0.25 to zero.

The plots show that the different configurations of weighting factors affect the percentage of buffer fullness for this example system. The percentage of buffer fullness is cumulative for the production sequence at a given data point, note that 100% is a full buffer for all products and 90% is the buffer threshold in the plots. The best case of all configurations results in a buffer fullness value of 90.4%, when $\alpha = 0.125$ and $\beta = \gamma = \varepsilon = \eta = 0.218$ for test case #1-A and also when $\gamma = 0.45$ and $\alpha = \beta = \varepsilon = \eta = 0.138$ for test case #3-A.

The lowest buffer fullness value without lookahead is 81.8% when α is the dominant weighting factor with a value of 1.0 and all other factors are zero. When lookahead is used, the lowest buffer fullness value ranges from 83.9-84.2% for test cases #1-B,#2-B, and #4-B. Perhaps this is caused because the time to crash and time to refill are very similar components of the goodness equation.

In all test cases, the lookahead provides approximately the same buffer fullness values than when not using the lookahead function. Note that the difference between the best case and the worst case is only 6.5% of the average buffer level. These results show that the output sequences for all the test cases are feasible sequences, but some sequences are slightly better than others.

In some cases the system parameters will dominate the output of the algorithm and the weighting factors will have no effect. Consider the three product system discussed in section 6.7.1 Three Product Production System. All five configurations of weighting factors, both with and without lookahead, result in the same buffer fullness value of 62.6%.

### 6.7.4 Additional Test Cases for Weighting Parameters

Many sequencing authors validate their results by using three problem sets with various product demands for 5, 10, and 15 product production systems that were originally developed by Sumichrast and Russell [53]. The five product problem set was considered for this example, which has the product demand shown in the table below. The production system has an initial buffer level = 100% of maximum buffer levels. Two buffer threshold levels test cases were considered; threshold test case #1 has a buffer threshold of 95% and threshold test case #2 has a buffer threshold of 50%. Two changeover cost families, shown below, were considered to gain a

better understanding of the behavior of the weighting parameters. Note that A=2 and B=5 in the changeover cost family tables. The lookahead time was also varied in this example between 0 for lookahead test case #1 and 20 time units for lookahead test case #2.

Table 6.1: Product Demand, $UR(i)$

| Demand Case | Prod 1 | Prod 2 | Prod 3 | Prod 4 | Prod 5 |
|---|---|---|---|---|---|
| 1 | 15 | 2 | 1 | 1 | 1 |
| 2 | 10 | 5 | 2 | 2 | 1 |
| 3 | 6 | 6 | 5 | 2 | 1 |
| 4 | 4 | 4 | 4 | 4 | 3 |

Table 6.2: Changeover Cost Family #1

| C/O Family 1 | Prod 1 | Prod 2 | Prod 3 | Prod 4 | Prod 5 |
|---|---|---|---|---|---|
| Prod 1 | 1 | A | B | B | B |
| Prod 2 | A | 1 | B | B | B |
| Prod 3 | B | B | 1 | A | A |
| Prod 4 | B | B | A | 1 | A |
| Prod 5 | B | B | A | A | 1 |

Table 6.3: Changeover Cost Family #2

| C/O Family 2 | Prod 1 | Prod 2 | Prod 3 | Prod 4 | Prod 5 |
|---|---|---|---|---|---|
| Prod 1 | 1 | B | B | B | A |
| Prod 2 | B | 1 | A | A | B |
| Prod 3 | B | A | 1 | A | B |
| Prod 4 | B | A | A | 1 | B |
| Prod 5 | A | B | B | B | 1 |

Table 6.4: Weighting Factor Test Cases

| Wt Factor Case | $\alpha$ | $\beta$ | $\gamma$ | $\varepsilon$ | $\eta$ |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 |
| 6 | 1/3 | 0 | 0 | 1/3 | 1/3 |

The production rate and maximum buffer level are functions of the usage rate as follows:

$$PR(i) = \begin{bmatrix} 35 \times UR(i) & 30 \times UR(i) & 25 \times UR(i) & 20 \times UR(i) & 20 \times UR(i) \end{bmatrix}$$

$$BF_{\max}(i) = \begin{bmatrix} 5 \times PR(i) & 4 \times PR(i) & 3 \times PR(i) & 2 \times PR(i) & 2 \times PR(i) \end{bmatrix}$$

The results for the various test cases yield interesting results for these test cases. The use of lookahead prevents the simulated production run from crashing but the use of lookahead causes an increase in the maximum percent of buffer level swing. Note that the percentage of buffer level swing is measured by subtracting the lowest percent of buffer fullness from one for the 100 steps of the simulated sequence. Therefore the lookahead function produces a sequence with more fluctuation in the buffer levels but no product buffers crash during the sequence. Note that all the results tables are sorted first by buffer threshold test case, secondly by maximum buffer level swing, and finally by percent of buffer fullness. Note that the highlighted rows are cases that crashed.

Table 6.5: Results for Demand Test Case #1, C/O Family #1, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 2 | 87.1 | 52.1 | 79.8 | 406.2 | 1 | 2 |
| 3 | 83.0 | 60.5 | 79.9 | 533.1 | 1 | 3 |
| 99 | 77.6 | 76.9 | 79.9 | 619.7 | 2 | 3 |
| 1 | 77.2 | 79.4 | 81.6 | 268.3 | 1 | 1 |
| 98 | 76.7 | 90.2 | 79.8 | 567.7 | 2 | 2 |
| 6 | 76.8 | 91.6 | 79.8 | 266.9 | 1 | 6 |
| 100 | 78.5 | 95.6 | 79.6 | 267.5 | 2 | 4 |
| 97 | 73.0 | 99.7 | 81.0 | 270.4 | 2 | 1 |
| 101 | 76.1 | 99.8 | 80.1 | 479.5 | 2 | 5 |
| 102 | 70.2 | 99.8 | 80.8 | 270.9 | 2 | 6 |
| 4 | 39.9 | 101.4 | 84.6 | 121.7 | 1 | 4 |
| 5 | 15.4 | 108.5 | 82.9 | 43.4 | 1 | 5 |

Table 6.6: Results for Demand Test Case #1, C/O Family #1, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 7 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 1 |
| 12 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 6 |
| 108 | 72.3 | 74.2 | 35.9 | 551.6 | 2 | 6 |
| 103 | 72.1 | 75.8 | 35.9 | 548.4 | 2 | 1 |
| 9 | 72.3 | 76.4 | 36.6 | 538.3 | 1 | 3 |
| 11 | 72.2 | 78.3 | 37.0 | 564.5 | 1 | 5 |
| 104 | 72.3 | 79.5 | 38.3 | 561.2 | 2 | 2 |
| 105 | 71.6 | 82.0 | 37.3 | 581.2 | 2 | 3 |
| 106 | 72.3 | 82.1 | 35.9 | 551.6 | 2 | 4 |
| 10 | 72.3 | 83.6 | 34.0 | 544.8 | 1 | 4 |
| 107 | 71.9 | 96.5 | 36.4 | 574.7 | 2 | 5 |
| 8 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 2 |

Table 6.7: Results for Demand Test Case #1, C/O Family #2, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 15 | 87.0 | 51.4 | 79.7 | 410.3 | 1 | 3 |
| 14 | 86.1 | 51.9 | 79.7 | 455.3 | 1 | 2 |
| 18 | 81.0 | 68.4 | 79.8 | 387.3 | 1 | 6 |
| 16 | 86.3 | 69.8 | 79.6 | 388.0 | 1 | 4 |
| 112 | 86.3 | 69.8 | 79.6 | 388.0 | 2 | 4 |
| 114 | 84.2 | 71.9 | 79.8 | 342.3 | 2 | 6 |
| 111 | 79.5 | 77.4 | 79.9 | 619.8 | 2 | 3 |
| 13 | 73.8 | 86.0 | 80.5 | 548.1 | 1 | 1 |
| 110 | 76.2 | 90.5 | 80.0 | 570.4 | 2 | 2 |
| 109 | 71.2 | 96.1 | 80.5 | 603.4 | 2 | 1 |
| 113 | 78.4 | 99.98 | 80.0 | 461.3 | 2 | 5 |
| 17 | 8.9 | 105.8 | 82.7 | 42.3 | 1 | 5 |

Table 6.8: Results for Demand Test Case #1, C/O Family #2, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 24 | 73.0 | 65.4 | 45.1 | 542.8 | 1 | 6 |
| 19 | 72.5 | 65.8 | 46.3 | 563.7 | 1 | 1 |
| 120 | 72.8 | 69.1 | 45.2 | 537.7 | 2 | 6 |
| 21 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 3 |
| 23 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 5 |
| 116 | 72.0 | 86.0 | 46.6 | 568.6 | 2 | 2 |
| 22 | 72.0 | 86.2 | 45.7 | 578.2 | 1 | 4 |
| 118 | 72.4 | 86.2 | 44.9 | 563.4 | 2 | 4 |
| 20 | 72.2 | 88.0 | 45.7 | 562.1 | 1 | 2 |
| 119 | 72.2 | 88.0 | 45.7 | 562.1 | 2 | 5 |
| 115 | 71.2 | 89.7 | 46.4 | 625.4 | 2 | 1 |
| 117 | 71.9 | 97.2 | 46.1 | 585.3 | 2 | 3 |

Table 6.9: Results for Demand Test Case #2, C/O Family #1, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 26 | 87.1 | 52.1 | 79.8 | 406.2 | 1 | 2 |
| 27 | 83.0 | 60.5 | 79.9 | 533.1 | 1 | 3 |
| 123 | 77.6 | 76.9 | 79.9 | 619.7 | 2 | 3 |
| 25 | 77.2 | 79.4 | 81.6 | 268.3 | 1 | 1 |
| 122 | 76.7 | 90.2 | 79.8 | 567.7 | 2 | 2 |
| 30 | 76.8 | 91.6 | 79.8 | 266.9 | 1 | 6 |
| 124 | 78.5 | 95.6 | 79.6 | 267.5 | 2 | 4 |
| 121 | 73.0 | 99.7 | 81.0 | 270.4 | 2 | 1 |
| 125 | 74.7 | 99.8 | 80.0 | 487.2 | 2 | 5 |
| 126 | 70.2 | 99.8 | 80.8 | 270.9 | 2 | 6 |
| 28 | 39.9 | 101.4 | 84.6 | 121.7 | 1 | 4 |
| 29 | 15.1 | 117.2 | 83.2 | 46.9 | 1 | 5 |

Table 6.10: Results for Demand Test Case #2, C/O Family #1, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 31 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 1 |
| 36 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 6 |
| 132 | 72.3 | 74.2 | 35.9 | 551.6 | 2 | 6 |
| 127 | 72.1 | 75.8 | 35.9 | 548.4 | 2 | 1 |
| 33 | 72.3 | 76.4 | 36.6 | 538.3 | 1 | 3 |
| 128 | 72.3 | 79.5 | 38.3 | 561.2 | 2 | 2 |
| 129 | 71.6 | 82.0 | 37.3 | 581.2 | 2 | 3 |
| 130 | 72.3 | 82.1 | 35.9 | 551.6 | 2 | 4 |
| 34 | 72.3 | 83.6 | 34.0 | 544.8 | 1 | 4 |
| 131 | 71.9 | 96.5 | 36.4 | 574.7 | 2 | 5 |
| 32 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 2 |
| 35 | 62.5 | 100.8 | 37.5 | 493.6 | 1 | 5 |

Table 6.11: Results for Demand Test Case #2, C/O Family #2, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 39 | 87.0 | 51.4 | 79.7 | 410.3 | 1 | 3 |
| 38 | 86.1 | 51.9 | 79.7 | 455.3 | 1 | 2 |
| 42 | 81.0 | 68.4 | 79.8 | 387.3 | 1 | 6 |
| 40 | 86.3 | 69.8 | 79.6 | 388.0 | 1 | 4 |
| 136 | 86.3 | 69.8 | 79.6 | 388.0 | 2 | 4 |
| 138 | 84.2 | 71.9 | 79.8 | 342.3 | 2 | 6 |
| 135 | 79.5 | 77.4 | 79.9 | 619.8 | 2 | 3 |
| 37 | 73.8 | 86.0 | 80.5 | 548.1 | 1 | 1 |
| 134 | 76.2 | 90.5 | 80.0 | 570.4 | 2 | 2 |
| 133 | 71.2 | 96.1 | 80.5 | 603.4 | 2 | 1 |
| 137 | 78.5 | 99.98 | 79.9 | 461.8 | 2 | 5 |
| 41 | 8.2 | 100.2 | 82.3 | 40.1 | 1 | 5 |

Table 6.12: Results for Demand Test Case #2, C/O Family #2, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 48 | 73.0 | 65.4 | 45.1 | 542.8 | 1 | 6 |
| 43 | 72.5 | 65.8 | 46.3 | 563.7 | 1 | 1 |
| 144 | 72.8 | 69.1 | 45.2 | 537.7 | 2 | 6 |
| 45 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 3 |
| 47 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 5 |
| 140 | 72.0 | 86.0 | 46.6 | 568.6 | 2 | 2 |
| 46 | 72.0 | 86.2 | 45.7 | 578.2 | 1 | 4 |
| 142 | 72.4 | 86.2 | 44.9 | 563.4 | 2 | 4 |
| 44 | 72.2 | 88.0 | 45.7 | 562.1 | 1 | 2 |
| 143 | 72.2 | 88.0 | 45.7 | 562.1 | 2 | 5 |
| 139 | 71.2 | 89.7 | 46.4 | 625.4 | 2 | 1 |
| 141 | 71.9 | 97.2 | 46.1 | 585.3 | 2 | 3 |

Table 6.13: Results for Demand Test Case #3, C/O Family #1, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 50 | 87.1 | 52.1 | 79.8 | 406.2 | 1 | 2 |
| 51 | 83.0 | 60.5 | 79.9 | 533.1 | 1 | 3 |
| 147 | 77.6 | 76.9 | 79.9 | 619.7 | 2 | 3 |
| 49 | 77.2 | 79.4 | 81.6 | 268.3 | 1 | 1 |
| 146 | 76.7 | 90.2 | 79.8 | 567.7 | 2 | 2 |
| 54 | 76.8 | 91.6 | 79.8 | 266.9 | 1 | 6 |
| 149 | 74.2 | 94.8 | 80.0 | 502.2 | 2 | 5 |
| 148 | 78.5 | 95.6 | 79.6 | 267.5 | 2 | 4 |
| 145 | 73.0 | 99.7 | 81.0 | 270.4 | 2 | 1 |
| 150 | 70.2 | 99.8 | 80.8 | 270.9 | 2 | 6 |
| 52 | 39.9 | 101.4 | 84.6 | 121.7 | 1 | 4 |
| 53 | 18.9 | 112.4 | 83.9 | 54.8 | 1 | 5 |

Table 6.14: Results for Demand Test Case #3, C/O Family #1, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 55 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 1 |
| 60 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 6 |
| 156 | 72.3 | 74.2 | 35.9 | 551.6 | 2 | 6 |
| 151 | 72.1 | 75.8 | 35.9 | 548.4 | 2 | 1 |
| 57 | 72.3 | 76.4 | 36.6 | 538.3 | 1 | 3 |
| 152 | 72.3 | 79.5 | 38.3 | 561.2 | 2 | 2 |
| 153 | 71.6 | 82.0 | 37.3 | 581.2 | 2 | 3 |
| 154 | 72.3 | 82.1 | 35.9 | 551.6 | 2 | 4 |
| 58 | 72.3 | 83.6 | 34.0 | 544.8 | 1 | 4 |
| 155 | 71.9 | 96.5 | 36.4 | 574.7 | 2 | 5 |
| 56 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 2 |
| 59 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 5 |

Table 6.15: Results for Demand Test Case #3, C/O Family #2, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 63 | 87.0 | 51.4 | 79.7 | 410.3 | 1 | 3 |
| 62 | 86.1 | 51.9 | 79.7 | 455.3 | 1 | 2 |
| 66 | 81.0 | 68.4 | 79.8 | 387.3 | 1 | 6 |
| 64 | 86.3 | 69.8 | 79.6 | 388.0 | 1 | 4 |
| 160 | 86.3 | 69.8 | 79.6 | 388.0 | 2 | 4 |
| 162 | 84.2 | 71.9 | 79.8 | 342.3 | 2 | 6 |
| 159 | 79.5 | 77.4 | 79.9 | 619.8 | 2 | 3 |
| 61 | 73.8 | 86.0 | 80.5 | 548.1 | 1 | 1 |
| 158 | 76.2 | 90.5 | 80.0 | 570.4 | 2 | 2 |
| 157 | 71.2 | 96.1 | 80.5 | 603.4 | 2 | 1 |
| 161 | 79.4 | 99.96 | 79.7 | 444.1 | 2 | 5 |
| 65 | 11.5 | 110.1 | 82.4 | 57.0 | 1 | 5 |

Table 6.16: Results for Demand Test Case #3, C/O Family #2, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 72 | 73.0 | 65.4 | 45.1 | 542.8 | 1 | 6 |
| 67 | 72.5 | 65.8 | 46.3 | 563.7 | 1 | 1 |
| 168 | 72.8 | 69.1 | 45.2 | 537.7 | 2 | 6 |
| 69 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 3 |
| 71 | 71.5 | 83.8 | 46.4 | 593.3 | 1 | 5 |
| 164 | 72.0 | 86.0 | 46.6 | 568.6 | 2 | 2 |
| 70 | 72.0 | 86.2 | 45.7 | 578.2 | 1 | 4 |
| 166 | 72.4 | 86.2 | 44.9 | 563.4 | 2 | 4 |
| 68 | 72.2 | 88.0 | 45.7 | 562.1 | 1 | 2 |
| 167 | 72.2 | 88.0 | 45.7 | 562.1 | 2 | 5 |
| 163 | 71.2 | 89.7 | 46.4 | 625.4 | 2 | 1 |
| 165 | 71.9 | 97.2 | 46.1 | 585.3 | 2 | 3 |

Table 6.17: Results for Demand Test Case #4, C/O Family #1, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 74 | 87.1 | 52.1 | 79.8 | 406.2 | 1 | 2 |
| 75 | 83.0 | 60.5 | 79.9 | 533.1 | 1 | 3 |
| 171 | 77.6 | 76.9 | 79.9 | 619.7 | 2 | 3 |
| 73 | 77.2 | 79.4 | 81.6 | 268.3 | 1 | 1 |
| 173 | 82.2 | 81.0 | 79.7 | 470.8 | 2 | 5 |
| 78 | 76.8 | 91.6 | 79.8 | 266.9 | 1 | 6 |
| 172 | 78.5 | 95.6 | 79.6 | 267.5 | 2 | 4 |
| 170 | 76.7 | 99.5 | 79.8 | 567.7 | 2 | 2 |
| 169 | 73.0 | 99.7 | 81.0 | 270.4 | 2 | 1 |
| 174 | 70.2 | 99.8 | 80.8 | 270.9 | 2 | 6 |
| 76 | 39.9 | 101.4 | 84.6 | 121.7 | 1 | 4 |
| 77 | 13.8 | 101.7 | 83.6 | 40.7 | 1 | 5 |

Table 6.18: Results for Demand Test Case #4, C/O Family #1, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 79 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 1 |
| 84 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 6 |
| 180 | 72.3 | 74.2 | 35.9 | 551.6 | 2 | 6 |
| 175 | 72.1 | 75.8 | 35.9 | 548.4 | 2 | 1 |
| 81 | 72.3 | 76.4 | 36.6 | 538.3 | 1 | 3 |
| 83 | 72.2 | 78.3 | 37.0 | 564.5 | 1 | 5 |
| 176 | 72.3 | 79.5 | 38.3 | 561.2 | 2 | 2 |
| 177 | 71.6 | 82.0 | 37.3 | 581.2 | 2 | 3 |
| 178 | 72.3 | 82.1 | 35.9 | 551.6 | 2 | 4 |
| 82 | 72.3 | 83.6 | 34.0 | 544.8 | 1 | 4 |
| 179 | 71.9 | 96.5 | 36.4 | 574.7 | 2 | 5 |
| 80 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 2 |

Table 6.19: Results for Demand Test Case #4, C/O Family #2, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 87 | 87.0 | 51.4 | 79.7 | 410.3 | 1 | 3 |
| 86 | 86.1 | 51.9 | 79.7 | 455.3 | 1 | 2 |
| 90 | 81.0 | 68.4 | 79.8 | 387.3 | 1 | 6 |
| 88 | 86.3 | 69.8 | 79.6 | 388.0 | 1 | 4 |
| 184 | 86.3 | 69.8 | 79.6 | 388.0 | 2 | 4 |
| 186 | 84.2 | 71.9 | 79.8 | 342.3 | 2 | 6 |
| 185 | 83.3 | 76.1 | 79.7 | 436.5 | 2 | 5 |
| 183 | 79.5 | 77.4 | 79.9 | 619.8 | 2 | 3 |
| 85 | 73.8 | 86.0 | 80.5 | 548.1 | 1 | 1 |
| 182 | 76.2 | 90.5 | 80.0 | 570.4 | 2 | 2 |
| 181 | 71.2 | 96.1 | 80.5 | 603.4 | 2 | 1 |
| 89 | 15.9 | 102.4 | 80.6 | 88.1 | 1 | 5 |

Table 6.20: Results for Demand Test Case #4, C/O Family #2, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 96 | 73.0 | 65.4 | 45.1 | 542.8 | 1 | 6 |
| 91 | 72.5 | 65.8 | 46.3 | 563.7 | 1 | 1 |
| 192 | 72.8 | 69.1 | 45.2 | 537.7 | 2 | 6 |
| 93 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 3 |
| 188 | 72.0 | 86.0 | 46.6 | 568.6 | 2 | 2 |
| 94 | 72.0 | 86.2 | 45.7 | 578.2 | 1 | 4 |
| 190 | 72.4 | 86.2 | 44.9 | 563.4 | 2 | 4 |
| 92 | 72.2 | 88.0 | 45.7 | 562.1 | 1 | 2 |
| 191 | 72.2 | 88.0 | 45.7 | 562.1 | 2 | 5 |
| 187 | 71.2 | 89.7 | 46.4 | 625.4 | 2 | 1 |
| 189 | 71.9 | 97.2 | 46.1 | 585.3 | 2 | 3 |
| 95 | 15.5 | 100.5 | 46.3 | 148.9 | 1 | 5 |

#### 6.7.4.1 Alternative Lookahead Selection of Additional Test Cases

The results in this section are based on the same test cases that were presented in the previous section, with the only difference being the lookahead selection method. The following results were found using the alternative lookahead selection method in which the sequence with the largest minimum goodness value over the lookahead time is the "best" sequence. The first product of the "best "sequence is selected as the next product in the production sequence.

The alternative selection method yields interesting results for these test cases. The use of the alternative lookahead selection again prevents the simulated production from crashing. Note that the alternative lookahead selection method actually causes a decrease in the maximum percent of

buffer level swing. Therefore the alternative lookahead method produces a sequence with less fluctuation in the buffer levels and no product buffers crash during the sequence. Note that all the results tables are again sorted by buffer threshold test case, then by maximum swing, and then by percent of buffer fullness. Note that the highlighted rows are cases that crashed.

Table 6.21: Alt Method for Demand Case #1, C/O Family #1, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 98 | 86.4 | 37.2 | 79.8 | 436.4 | 2 | 2 |
| 99 | 86.4 | 37.2 | 79.8 | 436.4 | 2 | 3 |
| 2 | 87.1 | 52.1 | 79.8 | 406.2 | 1 | 2 |
| 3 | 83.0 | 60.5 | 79.9 | 533.1 | 1 | 3 |
| 1 | 77.2 | 79.4 | 81.6 | 268.3 | 1 | 1 |
| 97 | 76.9 | 79.6 | 81.4 | 269.0 | 2 | 1 |
| 6 | 76.8 | 91.6 | 79.8 | 266.9 | 1 | 6 |
| 100 | 78.5 | 95.6 | 79.6 | 267.5 | 2 | 4 |
| 102 | 74.3 | 99.4 | 81.6 | 257.2 | 2 | 6 |
| 101 | 76.1 | 99.8 | 80.1 | 479.5 | 2 | 5 |
| 4 | 39.9 | 101.4 | 84.6 | 121.7 | 1 | 4 |
| 5 | 15.4 | 108.5 | 82.9 | 43.4 | 1 | 5 |

Table 6.22: Alt Method for Demand Case #1, C/O Family #1, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 108 | 72.7 | 68.7 | 35.4 | 544.7 | 2 | 6 |
| 7 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 1 |
| 12 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 6 |
| 103 | 72.0 | 71.8 | 36.0 | 581.3 | 2 | 1 |
| 105 | 72.3 | 76.4 | 37.4 | 548.5 | 2 | 3 |
| 9 | 72.3 | 76.4 | 36.6 | 538.3 | 1 | 3 |
| 11 | 72.2 | 78.3 | 37.0 | 564.5 | 1 | 5 |
| 104 | 72.4 | 81.4 | 38.3 | 556.3 | 2 | 2 |
| 106 | 72.3 | 82.1 | 35.9 | 551.6 | 2 | 4 |
| 10 | 72.3 | 83.6 | 34.0 | 544.8 | 1 | 4 |
| 107 | 71.9 | 96.5 | 36.4 | 574.7 | 2 | 5 |
| 8 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 2 |

Table 6.23: Alt Method for Demand Case #1, C/O Family #2, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 15 | 87.0 | 51.4 | 79.7 | 410.3 | 1 | 3 |
| 14 | 86.1 | 51.9 | 79.7 | 455.3 | 1 | 2 |
| 111 | 82.9 | 53.0 | 79.8 | 545.1 | 2 | 3 |
| 110 | 83.3 | 56.9 | 79.9 | 540.9 | 2 | 2 |
| 18 | 81.0 | 68.4 | 79.8 | 387.3 | 1 | 6 |
| 16 | 86.3 | 69.8 | 79.6 | 388.0 | 1 | 4 |
| 112 | 86.3 | 69.8 | 79.6 | 388.0 | 2 | 4 |
| 114 | 80.8 | 69.8 | 80.0 | 423.5 | 2 | 6 |
| 109 | 72.2 | 84.3 | 80.7 | 583.4 | 2 | 1 |
| 13 | 73.8 | 86.0 | 80.5 | 548.1 | 1 | 1 |
| 113 | 78.4 | 99.98 | 80.0 | 461.3 | 2 | 5 |
| 17 | 8.9 | 105.8 | 82.7 | 42.3 | 1 | 5 |

Table 6.24: Alt Method for Demand Case #1, C/O Family #2, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 24 | 73.0 | 65.4 | 45.1 | 542.8 | 1 | 6 |
| 19 | 72.5 | 65.8 | 46.3 | 563.7 | 1 | 1 |
| 115 | 72.5 | 65.8 | 46.3 | 563.7 | 2 | 1 |
| 120 | 72.8 | 69.1 | 45.2 | 537.7 | 2 | 6 |
| 21 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 3 |
| 23 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 5 |
| 117 | 72.5 | 78.9 | 46.9 | 558.9 | 2 | 3 |
| 22 | 72.0 | 86.2 | 45.7 | 578.2 | 1 | 4 |
| 118 | 72.4 | 86.2 | 44.9 | 563.4 | 2 | 4 |
| 116 | 71.7 | 88.0 | 46.5 | 581.2 | 2 | 2 |
| 20 | 72.2 | 88.0 | 45.7 | 562.1 | 1 | 2 |
| 119 | 72.2 | 88.0 | 45.7 | 562.1 | 2 | 5 |

Table 6.25: Alt Method for Demand Case #2, C/O Family #1, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 122 | 86.4 | 37.2 | 79.8 | 436.4 | 2 | 2 |
| 123 | 86.4 | 37.2 | 79.8 | 436.4 | 2 | 3 |
| 26 | 87.1 | 52.1 | 79.8 | 406.2 | 1 | 2 |
| 27 | 83.0 | 60.5 | 79.9 | 533.1 | 1 | 3 |
| 25 | 77.2 | 79.4 | 81.6 | 268.3 | 1 | 1 |
| 121 | 76.9 | 79.6 | 81.4 | 269.0 | 2 | 1 |
| 30 | 76.8 | 91.6 | 79.8 | 266.9 | 1 | 6 |
| 124 | 78.5 | 95.6 | 79.6 | 267.5 | 2 | 4 |
| 126 | 74.3 | 99.4 | 81.6 | 257.2 | 2 | 6 |
| 125 | 74.7 | 99.8 | 80.0 | 487.2 | 2 | 5 |
| 28 | 39.9 | 101.4 | 84.6 | 121.7 | 1 | 4 |
| 29 | 15.1 | 117.2 | 83.2 | 46.9 | 1 | 5 |

Table 6.26: Alt Method for Demand Case #2, C/O Family #1, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 132 | 72.7 | 68.7 | 35.4 | 544.7 | 2 | 6 |
| 31 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 1 |
| 36 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 6 |
| 127 | 72.0 | 71.8 | 36.0 | 581.3 | 2 | 1 |
| 129 | 72.3 | 76.4 | 37.4 | 548.5 | 2 | 3 |
| 33 | 72.3 | 76.4 | 36.6 | 538.3 | 1 | 3 |
| 128 | 72.4 | 81.4 | 38.3 | 556.3 | 2 | 2 |
| 130 | 72.3 | 82.1 | 35.9 | 551.6 | 2 | 4 |
| 34 | 72.3 | 83.6 | 34.0 | 544.8 | 1 | 4 |
| 131 | 71.9 | 96.5 | 36.4 | 574.7 | 2 | 5 |
| 32 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 2 |
| 35 | 62.5 | 100.8 | 37.5 | 493.6 | 1 | 5 |

Table 6.27: Alt Method for Demand Case #2, C/O Family #2, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 39 | 87.0 | 51.4 | 79.7 | 410.3 | 1 | 3 |
| 38 | 86.1 | 51.9 | 79.7 | 455.3 | 1 | 2 |
| 135 | 82.9 | 53.0 | 79.8 | 545.1 | 2 | 3 |
| 134 | 83.3 | 56.9 | 79.9 | 540.9 | 2 | 2 |
| 42 | 81.0 | 68.4 | 79.8 | 387.3 | 1 | 6 |
| 40 | 86.3 | 69.8 | 79.6 | 388.0 | 1 | 4 |
| 136 | 86.3 | 69.8 | 79.6 | 388.0 | 2 | 4 |
| 138 | 80.8 | 69.8 | 80.0 | 423.5 | 2 | 6 |
| 133 | 72.2 | 84.3 | 80.7 | 583.4 | 2 | 1 |
| 37 | 73.8 | 86.0 | 80.5 | 548.1 | 1 | 1 |
| 137 | 78.5 | 99.98 | 79.9 | 461.8 | 2 | 5 |
| 41 | 8.2 | 100.2 | 82.3 | 40.1 | 1 | 5 |

Table 6.28: Alt Method for Demand Case #2, C/O Family #2, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 48 | 73.0 | 65.4 | 45.1 | 542.8 | 1 | 6 |
| 43 | 72.5 | 65.8 | 46.3 | 563.7 | 1 | 1 |
| 139 | 72.5 | 65.8 | 46.3 | 563.7 | 2 | 1 |
| 144 | 72.8 | 69.1 | 45.2 | 537.7 | 2 | 6 |
| 45 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 3 |
| 47 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 5 |
| 141 | 72.5 | 78.9 | 46.9 | 558.9 | 2 | 3 |
| 46 | 72.0 | 86.2 | 45.7 | 578.2 | 1 | 4 |
| 142 | 72.4 | 86.2 | 44.9 | 563.4 | 2 | 4 |
| 140 | 71.7 | 88.0 | 46.5 | 581.2 | 2 | 2 |
| 44 | 72.2 | 88.0 | 45.7 | 562.1 | 1 | 2 |
| 143 | 72.2 | 88.0 | 45.7 | 562.1 | 2 | 5 |

Table 6.29: Alt Method for Demand Case #3, C/O Family #1, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 146 | 86.4 | 37.2 | 79.8 | 436.4 | 2 | 2 |
| 147 | 86.4 | 37.2 | 79.8 | 436.4 | 2 | 3 |
| 50 | 87.1 | 52.1 | 79.8 | 406.2 | 1 | 2 |
| 51 | 83.0 | 60.5 | 79.9 | 533.1 | 1 | 3 |
| 49 | 77.2 | 79.4 | 81.6 | 268.3 | 1 | 1 |
| 145 | 76.9 | 79.6 | 81.4 | 269.0 | 2 | 1 |
| 54 | 76.8 | 91.6 | 79.8 | 266.9 | 1 | 6 |
| 149 | 74.2 | 94.8 | 80.0 | 502.2 | 2 | 5 |
| 148 | 78.5 | 95.6 | 79.6 | 267.5 | 2 | 4 |
| 150 | 74.3 | 99.4 | 81.6 | 257.2 | 2 | 6 |
| 52 | 39.9 | 101.4 | 84.6 | 121.7 | 1 | 4 |
| 53 | 18.9 | 112.4 | 83.9 | 54.8 | 1 | 5 |

Table 6.30: Alt Method for Demand Case #3, C/O Family #1, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 156 | 72.7 | 68.7 | 35.4 | 544.7 | 2 | 6 |
| 55 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 1 |
| 60 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 6 |
| 151 | 72.0 | 71.8 | 36.0 | 581.3 | 2 | 1 |
| 153 | 72.3 | 76.4 | 37.4 | 548.5 | 2 | 3 |
| 57 | 72.3 | 76.4 | 36.6 | 538.3 | 1 | 3 |
| 152 | 72.4 | 81.4 | 38.3 | 556.3 | 2 | 2 |
| 154 | 72.3 | 82.1 | 35.9 | 551.6 | 2 | 4 |
| 58 | 72.3 | 83.6 | 34.0 | 544.8 | 1 | 4 |
| 155 | 71.9 | 96.5 | 36.4 | 574.7 | 2 | 5 |
| 56 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 2 |
| 59 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 5 |

Table 6.31: Alt Method for Demand Case #3, C/O Family #2, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 63 | 87.0 | 51.4 | 79.7 | 410.3 | 1 | 3 |
| 62 | 86.1 | 51.9 | 79.7 | 455.3 | 1 | 2 |
| 159 | 82.9 | 53.0 | 79.8 | 545.1 | 2 | 3 |
| 158 | 83.3 | 56.9 | 79.9 | 540.9 | 2 | 2 |
| 66 | 81.0 | 68.4 | 79.8 | 387.3 | 1 | 6 |
| 64 | 86.3 | 69.8 | 79.6 | 388.0 | 1 | 4 |
| 160 | 86.3 | 69.8 | 79.6 | 388.0 | 2 | 4 |
| 162 | 80.8 | 69.8 | 80.0 | 423.5 | 2 | 6 |
| 157 | 72.2 | 84.3 | 80.7 | 583.4 | 2 | 1 |
| 61 | 73.8 | 86.0 | 80.5 | 548.1 | 1 | 1 |
| 161 | 79.4 | 99.96 | 79.7 | 444.1 | 2 | 5 |
| 65 | 11.5 | 110.1 | 82.4 | 57.0 | 1 | 5 |

Table 6.32: Alt Method for Demand Case #3, C/O Family #2, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 72 | 73.0 | 65.4 | 45.1 | 542.8 | 1 | 6 |
| 67 | 72.5 | 65.8 | 46.3 | 563.7 | 1 | 1 |
| 163 | 72.5 | 65.8 | 46.3 | 563.7 | 2 | 1 |
| 168 | 72.8 | 69.1 | 45.2 | 537.7 | 2 | 6 |
| 69 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 3 |
| 165 | 72.5 | 78.9 | 46.9 | 558.9 | 2 | 3 |
| 71 | 71.5 | 83.8 | 46.4 | 593.3 | 1 | 5 |
| 70 | 72.0 | 86.2 | 45.7 | 578.2 | 1 | 4 |
| 166 | 72.4 | 86.2 | 44.9 | 563.4 | 2 | 4 |
| 164 | 71.7 | 88.0 | 46.5 | 581.2 | 2 | 2 |
| 68 | 72.2 | 88.0 | 45.7 | 562.1 | 1 | 2 |
| 167 | 72.2 | 88.0 | 45.7 | 562.1 | 2 | 5 |

Table 6.33: Alt Method for Demand Case #4, C/O Family #1, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 170 | 86.4 | 37.2 | 79.8 | 436.4 | 2 | 2 |
| 171 | 86.4 | 37.2 | 79.8 | 436.4 | 2 | 3 |
| 74 | 87.1 | 52.1 | 79.8 | 406.2 | 1 | 2 |
| 75 | 83.0 | 60.5 | 79.9 | 533.1 | 1 | 3 |
| 73 | 77.2 | 79.4 | 81.6 | 268.3 | 1 | 1 |
| 169 | 76.9 | 79.6 | 81.4 | 269.0 | 2 | 1 |
| 173 | 82.2 | 81.0 | 79.7 | 470.8 | 2 | 5 |
| 78 | 76.8 | 91.6 | 79.8 | 266.9 | 1 | 6 |
| 172 | 78.5 | 95.6 | 79.6 | 267.5 | 2 | 4 |
| 174 | 74.3 | 99.4 | 81.6 | 257.2 | 2 | 6 |
| 76 | 39.9 | 101.4 | 84.6 | 121.7 | 1 | 4 |
| 77 | 13.8 | 101.7 | 83.6 | 40.7 | 1 | 5 |

Table 6.34: Alt Method for Demand Case #4, C/O Family #1, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 180 | 72.7 | 68.7 | 35.4 | 544.7 | 2 | 6 |
| 79 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 1 |
| 84 | 72.0 | 71.8 | 36.0 | 581.3 | 1 | 6 |
| 175 | 72.0 | 71.8 | 36.0 | 581.3 | 2 | 1 |
| 177 | 72.3 | 76.4 | 37.4 | 548.5 | 2 | 3 |
| 81 | 72.3 | 76.4 | 36.6 | 538.3 | 1 | 3 |
| 83 | 72.2 | 78.3 | 37.0 | 564.5 | 1 | 5 |
| 176 | 72.4 | 81.4 | 38.3 | 556.3 | 2 | 2 |
| 178 | 72.3 | 82.1 | 35.9 | 551.6 | 2 | 4 |
| 82 | 72.3 | 83.6 | 34.0 | 544.8 | 1 | 4 |
| 179 | 71.9 | 96.5 | 36.4 | 574.7 | 2 | 5 |
| 80 | 71.9 | 96.5 | 36.4 | 574.7 | 1 | 2 |

Table 6.35: Alt Method for Demand Case #4, C/O Family #2, and Threshold Case #1

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 86 | 86.1 | 51.9 | 79.7 | 455.3 | 1 | 2 |
| 183 | 82.9 | 53.0 | 79.8 | 545.1 | 2 | 3 |
| 182 | 83.3 | 56.9 | 79.9 | 540.9 | 2 | 2 |
| 90 | 81.0 | 68.4 | 79.8 | 387.3 | 1 | 6 |
| 88 | 86.3 | 69.8 | 79.6 | 388.0 | 1 | 4 |
| 184 | 86.3 | 69.8 | 79.6 | 388.0 | 2 | 4 |
| 186 | 80.8 | 69.8 | 80.0 | 423.5 | 2 | 6 |
| 185 | 83.3 | 76.1 | 79.7 | 436.5 | 2 | 5 |
| 181 | 72.2 | 84.3 | 80.7 | 583.4 | 2 | 1 |
| 85 | 73.8 | 86.0 | 80.5 | 548.1 | 1 | 1 |
| 89 | 15.9 | 102.4 | 80.6 | 88.1 | 1 | 5 |

Table 6.36: Alt Method for Demand Case #4, C/O Family #2, and Threshold Case #2

| Test Case # | % Buffer Level | % Max Swing | % Setup Time | Total time | Lookahead Case | Wght Factor Case |
|---|---|---|---|---|---|---|
| 96 | 73.0 | 65.4 | 45.1 | 542.8 | 1 | 6 |
| 91 | 72.5 | 65.8 | 46.3 | 563.7 | 1 | 1 |
| 187 | 72.5 | 65.8 | 46.3 | 563.7 | 2 | 1 |
| 192 | 72.8 | 69.1 | 45.2 | 537.7 | 2 | 6 |
| 188 | 72.5 | 74.7 | 45.3 | 543.5 | 2 | 2 |
| 93 | 72.5 | 78.9 | 46.9 | 558.9 | 1 | 3 |
| 189 | 72.5 | 78.9 | 46.9 | 558.9 | 2 | 3 |
| 94 | 72.0 | 86.2 | 45.7 | 578.2 | 1 | 4 |
| 190 | 72.4 | 86.2 | 44.9 | 563.4 | 2 | 4 |
| 92 | 72.2 | 88.0 | 45.7 | 562.1 | 1 | 2 |
| 191 | 72.2 | 88.0 | 45.7 | 562.1 | 2 | 5 |
| 95 | 15.5 | 100.5 | 46.3 | 148.9 | 1 | 5 |

### 6.7.5 Pattern Production

Consider again the same eight product production system discussed in the examples in section 6.7.3 Weighting Parameters. All production parameters are unchanged except the buffer threshold is set to 100%. This example will highlight the ability of the algorithm to generate a production sequence for pattern production. The usage rate variation term of the goodness equation is equal to one and all other terms are set to zero. This configuration yields the following production pattern (the sequence repeats apparently indefinitely) when there is no lookahead time: 1-2-3-5-6-1-2-3-7-8-1-2-3-1-2-3-7-8-1-2-3-1-2-3-7-8-1-2-3-4-1-2-3-7-8.

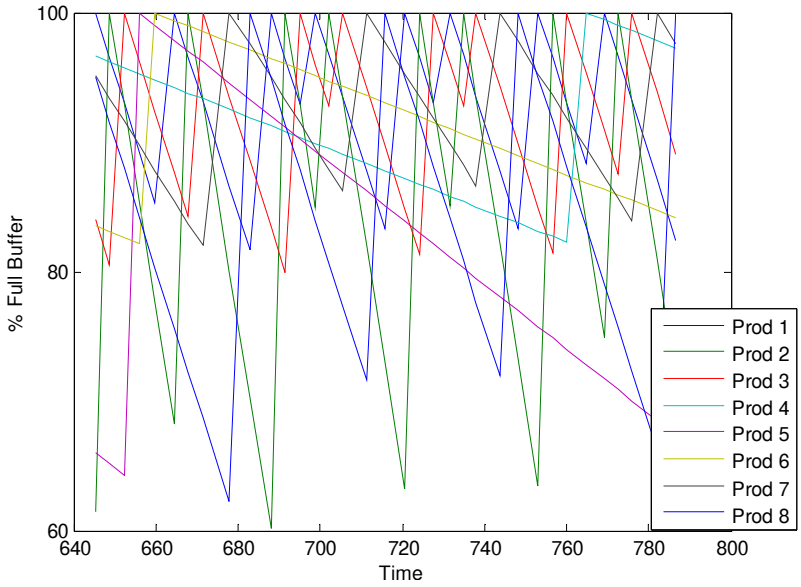


Figure 6.24: Eight Product System – Pattern Production

Other weighting factor combinations may yield different sequences depending upon what type of sequence is desired by the user. The algorithm will yield a sequence of 1-2-3-4-5-6-7-8 (the sequence repeats apparently indefinitely) when only the γ weighting factor for the time in queue term is used in the goodness equation.

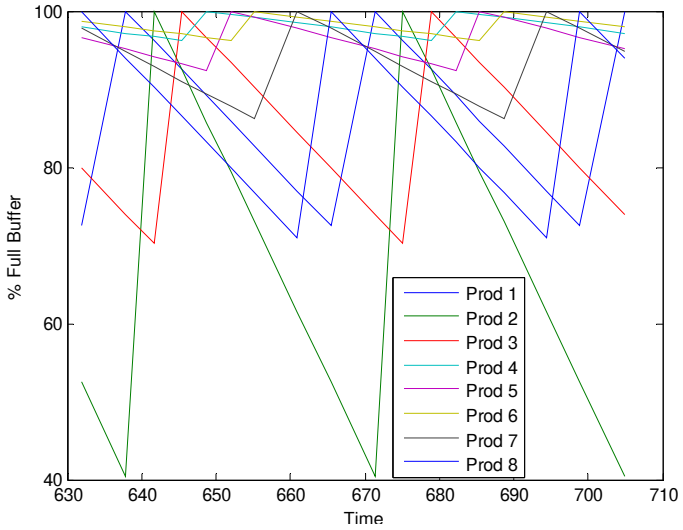


Figure 6.25: Pattern Production – γ = 1

# 7 Conclusions

This Chapter will summarize the work done and discuss the contributions of this dissertation as well as highlight areas that could benefit from future work.

## *7.1 Research Contributions*

This dissertation has addressed the problem of determining a product sequence for a Just-In-Time (JIT) production system when the process has significant sequence dependent setups. The issue of sequence dependent setups in JIT systems has only been published by a few researchers, but this is an important problem that needs to be resolved in order to help many manufacturers. The stability analysis conducted in this dissertation examines the difficulty of finding system parameters that are stable in an inherently unstable [5] production system. The analytical solution is general enough to be applied to a wide range of arc-node networks, but in this research the network is assumed to represent a production system with idle, setup, and refill nodes. The contribution of the analytical solution is that it shows that if a settled network has at least one node with a non-empty incoming and non-empty outgoing set, the network will have a stable trajectory that will cycle through the network.

The analytical stability results were further advanced in this dissertation by developing and implementing iterative method for determining the stability that attempts to find a set of stable regions for a user defined network. The method was demonstrated for two, three, and four-product production systems. If the stability analysis requires too much computational time, the sequencing algorithm can be used to determine a feasible product sequence for the production system.

This research contributes to JIT product sequencing algorithms by providing an algorithm that determines a feasible sequence for a system with sequence dependent setups. This sequencing algorithm also incorporates the ability to use lookahead to avoid the crashing of products in future steps when evaluating a product sequence. The lookahead feature increases the probability that the algorithm will find a feasible sequence by incorporating well researched lookahead philosophies with the emerging area of JIT sequencing with lookahead. This method can be easily implemented in production systems that have more products than the stability algorithm can analyze.

## *7.2 Summary of Stability Analysis*

The stability analysis method advances current research by considering a production system based on JIT manufacturing principles and includes sequence dependent setups between products and buffer threshold signals. The output regions were shown analytically to be stable for a settled arc-node network if one or more nodes have a non-empty incoming region and non-empty outgoing region. A product sequence trajectory through the network will remain stable at all future points in time if a point of the trajectory is contained within one of the output regions calculated by the algorithm. This is a significant result because it applies to any arc-node network with non-interrupted production and non-varying system parameters that refills each buffer completely. The algorithm requires very little computational time to determine output regions for two and three product production systems. The computational time required by the algorithm increases significantly with an increase in the number of products and number of arcs, or with an increase of multiple arcs attached to a given node.

The stability algorithm is intended to be an off-line tool that allows a user to determine whether or not a stable sequence trajectory exists for the system, based on the production system parameters. The algorithm could be used in conjunction with the product sequencing algorithm to verify that the given product sequence is a stable sequence. All future steps of the product sequence will remain stable if any one step of the sequence is located on an output region from the stability algorithm.

### 7.2.1 Future Work – Stability

The algorithm outputs a stable region in terms of a minimum and maximum value for each product in the production system. A two product system will have regions that are a one dimensional hyperrectangle which is a line between the two endpoints on the minimum values and the maximum values of each product in the region. A three product system will have regions that are a two-dimensional shape. As the number of products increase above three products, the regions become multi-dimensional hyperrectangles. The regions are always $k - 1$ dimensions, where $k$ is the number of products. Future work would be beneficial to evaluate output from the algorithm for systems with three or more products to gain a better understanding of the characteristics of the regions. For example, consider that a rectangle may be the proper shape of a three product region, while other cases could be more accurately represented by one or two triangles with a split region to remove instability from the middle of a rectangular region. Future work could develop a systematic approach to post process the output from the stability algorithm

to provide the most accurate stable regions for the given production system. This future work could also include a method to aid in visualization of the output regions because the multi-dimensional regions become very difficult to visualize.

The stability algorithm could be further refined to be more user friendly and robust by future work focusing on standard methods to troubleshoot non-converging solutions. Similarly to the sequencing algorithm, future work on this algorithm could develop recommendations for the user that could potentially yield a set of stable regions. These recommendations could start with increasing buffer size, increasing the production rate, decreasing usage rate, increasing buffer threshold, or reducing changeover costs.

Given that the computational time increases exponentially with an increase in the number of products and number of arcs, future work to improve the computational efficiency would be very beneficial. Analysis of the intersection, union, and cleanup functions would be a logical starting point for this work due to the perceived inefficiency and sheer volume of data that each function must process.

Incorporating the sequencing algorithm and stability together could improve robustness of the solutions as well as decrease computational time. The minimum and maximum buffer values from the sequencing algorithm could be used as the starting buffer regions for the stability algorithm to decrease the computational time required for the stability algorithm. Improved computational efficiency would allow production systems with a higher number of products to be analyzed by the algorithm.

Another potential area for future work is to allow the system parameters to vary over time and no longer require all parameters to be constant values. Variation in a production system is a real-world problem that is often unavoidable. Analysis of a system with variability would allow the stability algorithm to more accurately analyze most production systems.

A final possible area of future work is to join the stability analysis method human factor research to study the affect of buffer sizing upon worker efficiency and attitude as well as system stability. Large buffers are more likely to be stable but can cause workers to be less efficient and hide inherent problems in the production system. Research in this area could help to find the trade-offs between stability, buffer size, and the human factors.

## *7.3 Summary of Sequencing Algorithm*

The production sequencing algorithm developed in this research has proved to be an effective means of determining a production sequence for a JIT production system with significant sequence dependent setups. The algorithm provides a product sequence based on the given system parameters and user defined goodness weighting factors. The production sequence is not guaranteed to be an optimized product sequence for the production system, but it is a feasible sequence for the system over the lookahead period. The use of lookahead allows the algorithm to detect potential empty buffers and take corrective action prior to a product crashing.

The weighting factors used for calculating the goodness equation have some effect on the output of the algorithm as was show in the examples of output from the sequencing algorithms in Section 6.7.3 Weighting Parameters. A valid argument could also be made that the production system parameters play an equivalent or even more significant effect on the output of the algorithm. The examples showed that varying the different weighting parameters from zero to one changes the cumulative percentage of buffer fullness a very small amount. This fact highlights that the sequence is significantly influenced by the system parameters and not just by the goodness weighting factors. The three product example in Section 6.7.3.11 highlights a production system that is unaffected by the varying the weighting factors.

The intended use for the sequencing algorithm is as an on-line production sequencing tool on the manufacturing floor. The algorithm can be used to find a feasible production sequence in a real-world manufacturing environment or as a research tool to better understand behavior and implementation of a JIT system.

### 7.3.1 Future Work – Sequencing

The production sequencing algorithm is viable and beneficial to users in the current state but could be improved with additional future work. The weighting factors used for the current state goodness equation and lookahead goodness equation need to be further evaluated to better understand the affects of each factor. Further understanding of the weighting factor behavior would allow the user to define the weighting factors such that the sequence can be manipulated to provide a desired result. Perhaps investigation of a better normalizing calculation for each weighting factor would also increase the effectiveness of manipulating on the final sequence.

The sequencing algorithm currently will exit if any of the products crash during the sequence calculation. Additional future work to incorporate troubleshooting intelligence into the algorithm would be very beneficial to most users. This intelligence could provide methods or recommendations for the user of the algorithm to correct a crashed sequence. An experienced or knowledgeable user would not require this feature but it would be beneficial for novice users. A starting point for the troubleshooting could be for the algorithm to first identify the crashed product and then recommend possible solutions such as to increase lookahead time, increase buffer size, increase the production rate, decrease usage rate, increase buffer threshold, or reduce changeover costs. Depending upon the type of production system that is being modeled, it may not be possible to implement some of the recommended solutions, such as increasing the production rate or decreasing the usage rate. Often time the production capacity is a limiting factor and cannot be improved without a significant capital investment such as buying addition machines. The usage rate or customer demand is also often beyond the control of the manufacturer and if demand cannot be met then sales are lost [86]. Many manufacturers can benefit from implementing Lean principles to decrease setup and changeover cost which could aid the algorithm in being able to determine a non-crashing production sequence.

A final area of future work for the sequencing algorithm is the computational efficiency of the code used for the algorithm. The algorithm was developed in the midst of learning the programming language; therefore it is not eloquent or nor efficient. Reviewing the data handling methods could reveal a significant improvement in computational efficiency.

## Appendix I: Stability Algorithm Implementation

## Stability Algorithm:

```
clear all;
global num_of_prods,global setup,global U,global l,global l,global PR,global rho,global flag, global flag_length,global arc,global
node,global flag_ini,global num_of_arcs,global arcs, global message,global num_of_prods,global i,global flag,global cur_prod,global
node, global change_flag,global arcs_in,global arcs_out,global index_arc_i,global index_arc_j, global sheet_num_index,global
sheet_num_out_index,global limit_out_max,global limit_out_min,global limit_in_max,global limit_in_min,global
limit_out_max_new,global limit_out_min_new,global limit_in_max_new,global limit_in_min_new,global arc_star, global
arc_int,global flag,global flag_length,global star_flag,global cleanup_flag_y, global cleanup_flag_x,global arc_stored, global
arc_stored_index, global arc_intersected, global arc_int,global idle_flag,global sheet_index2,global flag_no_intersection

num_of_prods=3;
int_check_skip=0;
backward_stability=1;%set to 1 for checking sability backward propagation, 0 for forward check only
i=1;
flag_index=1;
initialize_stability(num_of_prods);
flag_ini=flag;
initialize_arc_limits(arcs,node,U,l,num_of_arcs,num_of_prods);
while (flag_index < 550) && (flag_length < 500) && (flag_length > 0)
   set_node_num(node);
   set_node_arcs(i,arcs,num_of_arcs);
   change_flag = 0;
   arc_stored_index=1;
   sheet_index2=1;
   arc_outer_loop_index=0;
   arc_int_outer_loop_index=0;
   for arc_out_index=1:index_arc_j%-1
      arc_outer_loop_index=arc_outer_loop_index+1;
      arc_inner_loop_index=0;
      arc_out=arcs_out(arc_out_index,:);
      arc_a = arc_out(1,1);
      arc_b = arc_out(1,2);
      arc_org(arc_out(1,1),arc_out(1,2)).limits = arc(arc_out(1,1),arc_out(1,2)).limits;
      arc_int(arc_a,arc_b).limits(num_of_prods,2,:)=[0];
      for sheet_num_out_index=1:size(arc(i,arc_out(1,2)).limits,3);
         arc_int_outer_loop_index=arc_int_outer_loop_index+1;
         arc_int_outer_sheet_index=sheet_num_out_index;
         %y/x_star is to be reset each time the following loops repeat
         for arc_in_index=1:index_arc_i%-1
            arc_in=arcs_in(arc_in_index,:);
            for sheet_num_index=1:size(arc(arc_in(1,1),i).limits,3);
               arc_inner_loop_index=arc_inner_loop_index+1;
               arc_inner_no_int=arcs_in(arc_in_index,:);
               arc_inner_no_int_index=sheet_num_index;
               set_current_limits(i,U,arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index);
               set_node_time(limit_in_max,limit_in_min,i,setup,cur_prod,U,PR,rho,arc,arc_in,arc_out,flag_index,
                     arc_inner_loop_index,sheet_num_out_index,sheet_num_index,l);
               for k=1:num_of_prods
                  calculate_new_out_limits(k,i,U,rho,l,node,arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,
                     sheet_num_out_index,limit_out_max,limit_out_min,limit_in_max,limit_in_min,idle_flag);
               end
               calculate_x_star(limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,k,i,U,rho,l,node,
                     arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_min,limit
                     _in_max,limit_in_min,num_of_prods,num_of_arcs,arcs);
               arc_int_check(arc_a,arc_b).limits=arc_star(arc_a,arc_b).limits(:,:,arc_inner_loop_index);
               if flag_index > int_check_skip
                  intersection_arc_int_check(arcs,num_of_arcs,limit_in_max_new,limit_in_min_new,limit_out_max_new,
                     limit_out_min_new,k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,li
                     mit_out_max,limit_out_min,limit_in_max,limit_in_min,num_of_prods,flag_index,arc_a,arc_b,sheet_index2,arc_inne
                     r_no_int,arc_inner_no_int_index,arc_int_check,arc_outer_loop_index,arc_int_outer_loop_index,arc_int_outer_sheet
                     _index);
               end
            end
         end
      end
      if size(arc_star(arc_a,arc_b).limits,2) > 0
         intersect_arc_star(arcs,num_of_arcs,limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,
            k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_mi
            n,limit_in_max,limit_in_min,num_of_prods,flag_index,arc_a,arc_b);
```

```
    end
    arc(arc_a,arc_b).limits=arc_int(arc_a,arc_b).limits;
    if size(arc(arc_a,arc_b).limits,3) >1
       cleanup_sheets(num_of_prods,arc_a,arc_b);
    end
    if size(arc(arc_a,arc_b).limits,3)>1
       union_arc(arc_int,arcs,num_of_arcs,limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,
            k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_mi
            n,limit_in_max,limit_in_min,num_of_prods,flag_index,arc_a,arc_b);
    end
    cleanup_sheets(num_of_prods,arc_a,arc_b);
    set_change_flag(arc,arc_org,num_of_prods,arc_a,arc_b,flag_index);
    update_change_flag_out(i,arc_in,arc_out,arcs_in,arcs_out);
    change_flag=0;
end
if flag_index > int_check_skip
    for arc_in_index=1:index_arc_i
       arc_in=arcs_in(arc_in_index,:);
       y=size(flag_no_intersection(arc_in(1,1),arc_in(1,2)).sheet,1);
       z=size(flag_no_intersection(arc_in(1,1),arc_in(1,2)).sheet,2);
       int_index=1;
       for int_index_row=1:y
          if min(flag_no_intersection(arc_in(1,1),arc_in(1,2)).sheet(int_index_row,:)) == 1
             arc(arc_in(1,1),arc_in(1,2)).limits(:,:,int_index_row)=0;
             change_flag=1;
             update_change_flag(i,arc_in,arc_out,arcs_in,arcs_out);
          end
       end
    end
    int_index_column=0;
    if backward_stability == 1
       for arc_out_index=1:index_arc_j
          arc_out=arcs_out(arc_out_index,:);
          z=size(arc_org(arc_out(1,1),arc_out(1,2)).limits,3);
          int_index=1;
          column_flag=[];
          while int_index <= z
             int_index_column=int_index_column+1;
             for arc_in_index=1:index_arc_i
                arc_in=arcs_in(arc_in_index,:);
                if min(flag_no_intersection(arc_in(1,1),arc_in(1,2)).sheet (:,int_index_column)) == 1
                   column_flag(1,arc_in_index)=1;
                else
                   column_flag(1,arc_in_index)=0;
                   arc_in_index=index_arc_i;
                   break
                end
             end
             if min(column_flag) == 1
                equal_flag=0;
                for sht_index=1:size(arc(arc_out(1,1),arc_out(1,2)).limits,3);
                   for prod_index=1:num_of_prods
                      if arc(arc_out(1,1),arc_out(1,2)).limits(k,:,sht_index)==arc_org(arc_out(1,1),arc_out(1,2)).limits(k,:,int_index);
                         equal_flag=equal_flag+1;
                      else
                         break
                      end
                   end
                   if equal_flag==num_of_prods
                      arc(arc_out(1,1),arc_out(1,2)).limits(:,:,sht_index)=0;
                      break
                   end
                   equal_flag=0;
                end
                change_flag=1;
                update_change_flag(i,arc_in,arc_out,arcs_in,arcs_out);
             end
             int_index=int_index+1;
             change_flag=0;
          end
```

```
      end
   end
end

flag_no_intersection(:,:)=[];
arc_outer_loop_index=0;
arc_int_outer_loop_index=0;
for arc_in_index=1:index_arc_i
   arc_outer_loop_index=arc_outer_loop_index+1;
   arc_in=arcs_in(arc_in_index,:);
   arc_inner_loop_index=0;
   arc_a = arc_in(1,1);
   arc_b = arc_in(1,2);
   arc_org(arc_in(1,1),arc_in(1,2)).limits = arc(arc_in(1,1),arc_in(1,2)).limits;
   arc_int(arc_a,arc_b).limits(num_of_prods,2,:)=[0];
   for sheet_num_index=1:size(arc(arc_in(1,1),i).limits,3);
      arc_int_outer_loop_index=arc_int_outer_loop_index+1;
      arc_int_outer_sheet_index=sheet_num_index;
      %x/y_star is cleared each time the following loop is repeated
      for arc_out_index=1:index_arc_j%-1
         arc_out=arcs_out(arc_out_index,:);
         for sheet_num_out_index=1:size(arc(i,arc_out(1,2)).limits,3);
            arc_inner_loop_index=arc_inner_loop_index+1;
            arc_inner_no_int=arcs_out(arc_out_index,:);
            arc_inner_no_int_index=sheet_num_out_index;
            set_current_limits(i,U,arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index);
            set_node_time(limit_in_max,limit_in_min,i,setup,cur_prod,U,PR,rho,arc,arc_in,arc_out,flag_index,
                arc_inner_loop_index,sheet_num_out_index,sheet_num_index,l);
            for k=1:num_of_prods
               calculate_new_in_limits(k,i,U,rho,l,node,arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,
                  sheet_num_out_index,limit_out_max,limit_out_min,limit_in_max,limit_in_min,idle_flag);
            end
            calculate_y_star(limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,k,i,U,rho,
                l,node,arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_mi
                n,limit_in_max,limit_in_min,num_of_prods,num_of_arcs,arcs);
            arc_int_check(arc_a,arc_b).limits=arc_star(arc_a,arc_b).limits(:,:,arc_inner_loop_index);
            if flag_index > int_check_skip
               intersection_arc_int_check(arcs,num_of_arcs,limit_in_max_new,limit_in_min_new,limit_out_max_new,
                  limit_out_min_new,k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,li
                  mit_out_max,limit_out_min,limit_in_max,limit_in_min,num_of_prods,flag_index,arc_a,arc_b,sheet_index2,arc_inne
                  r_no_int,arc_inner_no_int_index,arc_int_check,arc_outer_loop_index,arc_int_outer_loop_index,arc_int_outer_sheet
                  _index);
            end
         end
      end
   end
   if size(arc_star(arc_a,arc_b).limits,2) > 0
      intersect_arc_star(arcs,num_of_arcs,limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,
          k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_mi
          n,limit_in_max,limit_in_min,num_of_prods,flag_index,arc_a,arc_b);
   end
   arc(arc_a,arc_b).limits=arc_int(arc_a,arc_b).limits;
   if size(arc(arc_a,arc_b).limits,3) >1
      cleanup_sheets(num_of_prods,arc_a,arc_b);
   end
   if size(arc(arc_a,arc_b).limits,3)>1
      union_arc(arc_int,arcs,num_of_arcs,limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,
          k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_mi
          n,limit_in_max,limit_in_min,num_of_prods,flag_index,arc_a,arc_b);
   end
   cleanup_sheets(num_of_prods,arc_a,arc_b);
   set_change_flag(arc,arc_org,num_of_prods,arc_a,arc_b,flag_index);
   update_change_flag_in(i,arc_in,arc_out,arcs_in,arcs_out);
   change_flag=0;
end
if flag_index > int_check_skip
   if backward_stability == 1
      for arc_out_index=1:index_arc_j%-1
         arc_out=arcs_out(arc_out_index,:);
         y=size(flag_no_intersection(arc_out(1,1),arc_out(1,2)).sheet,1);
```

```
            z=size(flag_no_intersection(arc_out(1,1),arc_out(1,2)).sheet,2);
            int_index=1;
            for int_index_row=1:y
               if min(flag_no_intersection(arc_out(1,1),arc_out(1,2)).sheet(int_index_row,:)) == 1
                  arc(arc_out(1,1),arc_out(1,2)).limits(:,:,int_index_row)=0;
                  change_flag=1;
                  update_change_flag(i,arc_in,arc_out,arcs_in,arcs_out);
               end
            end
         end
      end
      int_index_column=0;
      for arc_in_index=1:index_arc_i%-1
         arc_in=arcs_in(arc_in_index,:);
         z=size(arc_org(arc_in(1,1),arc_in(1,2)).limits,3);
         int_index=1;
         column_flag=0;
         while int_index <= z
            int_index_column=int_index_column+1;
            for arc_out_index=1:index_arc_j
               arc_out=arcs_out(arc_out_index,:);
               if min(flag_no_intersection(arc_out(1,1),arc_out(1,2)).sheet(:,int_index_column)) == 1
                  column_flag(1,arc_out_index)=1;
               else
                  column_flag(1,arc_out_index)=0;
                  arc_out_index=index_arc_j;
                  break
               end
            end
            if (min(column_flag) == 1) && max(max(arc_org(arc_in(1,1),arc_in(1,2)).limits(:,:,int_index))) > 0
               equal_flag=0;
               for sht_index=1:size(arc(arc_in(1,1),arc_in(1,2)).limits,3)
                  for prod_index=1:num_of_prods
                     if arc(arc_in(1,1),arc_in(1,2)).limits(k,:,sht_index)==arc_org(arc_in(1,1),arc_in(1,2)).limits(k,:,int_index);
                        equal_flag=equal_flag+1;
                     else
                        break
                     end
                  end
                  if equal_flag==num_of_prods
                     arc(arc_in(1,1),arc_in(1,2)).limits(:,:,sht_index)=0;
                     break
                  end
                  equal_flag=0;
               end
               change_flag=1;
               update_change_flag(i,arc_in,arc_out,arcs_in,arcs_out);
            end
            change_flag=0;
            int_index=int_index+1;
         end
      end
   end
   flag_no_intersection(:,:)=[];
   flag_index=flag_index+1;
   update_node_number(i);
end
for i=1:num_of_arcs
   for j = 1:num_of_arcs
      if arcs(i,j) == 1
         arc_stored(i,j).limits=arc(i,j).limits;
         disp([i,j;])
         disp(round(arc(i,j).limits))
      end
   end
end
```

## initialize_stability Function:

```
function [arcs,arc,node,U,l,setup,PR,rho,flag,flag_length,num_of_arcs,message,i,flag_no_intersection]
      =initialize_stability(num_of_prods);
global setup,global U,global l,global l,global PR,global rho,global flag,global flag_length,global arc,global node,global
num_of_arcs,global arcs,global message
% define node type -- 1=idle, 2=setup, 3=fill
node(1).type=1; node(2).type=2;node(3).type=3; node(4).type=2;node(5).type=3; node(6).type=2;node(7).type=3;
% define the prod number for each node
node(2).product_num=1; node(3).product_num=1; node(4).product_num=2; node(5).product_num=2; node(6).product_num=3;
node(7).product_num=3;
setup=[5;5;5;5;5;5;5;5;5;];%setup cost (from,to)
U=[100;100;100;];%upper limit of product(i)
l=[99.9;99.9;99.9;];%lower limit of product(i)
PR=[30;30;30;];%production rate of product(i)
rho=[3;3;3;];%usage rate of product(i)
flag=[1 2 3 4 5 6 7 ;];
% three prods going thur idle or not
arcs=[0 1 0 1 0 1 0;%1 idle
     0 0 1 0 0 0 0;%2 setup1
     1 0 0 1 0 1 0;%3 fill 1
     0 0 0 0 1 0 0;%4 setup 2
     1 1 0 0 0 1 0;%5 fill 2
     0 0 0 0 0 0 1;%6 setup 3
     1 1 0 1 0 0 0;];%7 fill 3 %the connections of the network (from,to) for
flag_length=length(flag);
num_of_arcs=length(arcs);
message=0;
arc_index=1;
for i=1:num_of_arcs
   for j=1:num_of_arcs
      if arcs(i,j)==1
         for k=1:num_of_prods
         arc(i,j).limits(k,:)=[0 U(k);];
         end
      end
   end
end
flag_no_intersection(i,j).sheet(1)=0;
i=0;
```

## initialize_arc_limits Function:

```
function [arc]=initialize_arc_limits(arcs,node,U,l,num_of_arcs,num_of_prods);
global arc
for i=1:num_of_arcs
   for j=1:num_of_arcs
      if arcs(i,j)==1
         node_type_in=node(j).type;
         node_type_out=node(i).type;
         out_product=node(i).product_num;
         in_product=node(j).product_num;
         for k=1:num_of_prods
            %for setup node and product being setup, set max=l(k)
                        if node_type_in == 2 && in_product == k
               arc(i,j).limits(k,2)=l(k);
            end
            %for entering into idle node, min value l(k) for all k
                        if node_type_in == 1
               arc(i,j).limits(k,1)=l(k);
            end
            %outgoing arc constraints
            if node_type_out == 1 %for idle node, set minimum out going value at l(k)
               arc(i,j).limits(k,1)=l(k);
            end
            %for fill node and product being filled, set min=max=U(k)
                        if node_type_out == 3 && out_product == k
               arc(i,j).limits(k,:)=[U(k) U(k);];
```

```
            end
          end
        end
    end
end
```

## set_node_num Function:

```
function [i,cur_prod,flag]=set_node_num(node);
global i, global flag
i=flag(1);
if node(i).type ~= 1
   global cur_prod
   cur_prod=node(i).product_num;
end
```

## set_node_arcs Function:

```
function [arcs_in,arcs_out,index_arc_i,index_arc_j,sheet_num_index]=set_node_arcs(i,arcs,num_of_arcs)
global arcs_in,global arcs_out,global index_arc_i,global index_arc_j,global sheet_num_index
arcs_out=[];
arcs_in=[];
index_arc_i=1;
index_arc_j=1;
%define all the possile in and out arcs for a given node.
for j = 1:num_of_arcs
   if arcs(i,j) == 1
      arcs_out(index_arc_i,:)= [i,j;];
      index_arc_i=index_arc_i+1;
   end
   if arcs(j,i) == 1
      arcs_in(index_arc_j,:) = [j,i;];
      index_arc_j=index_arc_j+1;
   end
end
sheet_num_index=1;
in_size=size(arcs_in);
out_size=size(arcs_out);
index_arc_i=in_size(1);
index_arc_j=out_size(1);
```

## set_current_limits Function:

```
function[limit_out_max,limit_out_min,limit_in_max,limit_in_min,limit_out_max_new,limit_out_min_new,limit_in_max_new,
      limit_in_min_new]=set_current_limits(i,U,arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index)
global limit_out_max,global limit_out_min,global limit_in_max,global limit_in_min, global limit_out_max_new,global
limit_out_min_new,global limit_in_max_new,globallimit_in_min_new
%this defines the current limit values for in and out, min and max
limit_in_max=arc(arc_in(1,1),i).limits(:,2,sheet_num_index); limit_in_min=arc(arc_in(1,1),i).limits(:,1,sheet_num_index);
limit_out_max=arc(i,arc_out(1,2)).limits(:,2,sheet_num_out_index);
limit_out_min=arc(i,arc_out(1,2)).limits(:,1,sheet_num_out_index); limit_out_max_new=limit_out_max(:);
limit_out_min_new=limit_out_min(:);
limit_in_max_new=limit_in_max(:);
limit_in_min_new=limit_in_min(:);
```

## set_node_time Function:

```
function [node]=set_node_time(limit_in_max,limit_in_min,i,setup,cur_prod,U,PR,rho,arc,arc_in,arc_out,flag_index,
      arc_inner_loop_index,sheet_num_out_index,sheet_num_index,l);
global node, global num_of_prods
```

```
%this group of if statements are to calculate the amount of time required to pass through the node
if node(i).type == 1
   node(i).time_delta_max=(arc(arc_in(1,1),arc_in(1,2)).limits(node(arc_out(1,2)).product_num,2,sheet_num_index)
      -l(node(arc_out(1,2)).product_num))/rho(node(arc_out(1,2)).product_num);
   node(i).time_delta_min=(arc(arc_in(1,1),arc_in(1,2)).limits(node(arc_out(1,2)).product_num,1,sheet_num_index)
      -l(node(arc_out(1,2)).product_num))/rho(node(arc_out(1,2)).product_num);;
end
if node(i).type == 2
   node(i).time_delta=setup(cur_prod);
end
if node(i).type == 3
   node(i).time_delta_max=(U(cur_prod)-arc(arc_in(1),arc_in(2)).limits(cur_prod,1,sheet_num_index))/(PR(cur_prod)
      -rho(cur_prod));
   node(i).time_delta_min=(U(cur_prod)-arc(arc_in(1),arc_in(2)).limits(cur_prod,2,sheet_num_index))/(PR(cur_prod)-rho(cur_prod));
end
```

## calculate_new_out_limits Function:

```
function[limit_out_max_new,limit_out_min_new]=calculate_new_out_limits(k,i,U,rho,l,node,arc,arc_in,arc_out,arc_inner_loop_
      index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_min,limit_in_max,limit_in_min,idle_flag)
global limit_out_max_new,global limit_out_min_new
if node(i).type == 1 %for idle node
   if node(arc_out(1,2)).product_num ~= k
      idle_time_min=max(0,node(i).time_delta_min);
      idle_time_max=max(0,node(i).time_delta_max);
      limit_out_max_new(k)=limit_in_max(k)-idle_time_min*rho(k);;
      limit_out_min_new(k)=min(limit_in_min(k)-idle_time_min*rho(k),limit_in_max(k)-idle_time_max*rho(k));
   end
end
if node(i).type == 2 %for setup node
   limit_out_min_new(k)=limit_in_min(k) - node(i).time_delta*rho(k) ;
   limit_out_max_new(k)=limit_in_max(k) - node(i).time_delta*rho(k) ;
end
if node(i).type == 3 %for fill node
   if node(i).product_num ~= k
      limit_out_min_new(k)=limit_in_min(k) - node(i).time_delta_max*rho(k);
      limit_out_max_new(k)=limit_in_max(k) - node(i).time_delta_min*rho(k);
   end
end
```

## calculate_x_star Function:

```
function[arc_star,change_flag]=calculate_x_star(limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,k,i,
      U,rho,l,node,arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_min,lim
      it_in_max,limit_in_min,num_of_prods,num_of_arcs,arcs);
global arc_star, global change_flag
for o=1:num_of_prods
   arc_star(arc_out(1),arc_out(2)).limits(o,:,arc_inner_loop_index)=[0 0;];
 end
for k=1:num_of_prods
   arc_star(arc_out(1,1),arc_out(1,2)).limits(k,2,arc_inner_loop_index)=limit_out_max_new(k);
   arc_star(arc_out(1,1),arc_out(1,2)).limits(k,1,arc_inner_loop_index)=limit_out_min_new(k);
end
flag_equal=[];
a=arc_out(1,1);b=arc_out(1,2);
if size(arc_star(a,b).limits,3) >= 1
   z=size(arc_star(a,b).limits,3);
   m=arc_inner_loop_index;
   while m <= z
      for k=1:num_of_prods
         if (arc_star(a,b).limits(k,1,m)) == (arc_star(a,b).limits(k,2,m))
            flag_equal(k,1)=1;
         else
            flag_equal(k,1)=0;
         end
```

```
      end
      if min(min(flag_equal(:,:))) == 1
         arc_star(a,b).limits(:,:,m)=0;
      end
      flag_equal=[];
      m=m+1;
   end
end
flag_equal=[];
a=arc_out(1,1);b=arc_out(1,2);
if size(arc_star(a,b).limits,3) >= 1
   z=size(arc_star(a,b).limits,3);
   m=arc_inner_loop_index;
   while m <= z
      for k=1:num_of_prods
         if (arc_star(a,b).limits(k,1,m)) > (arc_star(a,b).limits(k,2,m))
            flag_equal(k,1)=1;
         else
            flag_equal(k,1)=0;
         end
      end
      if max(min(flag_equal(:,:))) == 1
         arc_star(a,b).limits(:,:,m)=0;
      end
      flag_equal=[];
      m=m+1;
   end
end
```

## intersection_arc_int_check Function:

```
function[arc,arc_int_check,cleanup_flag,arc_int,flag_no_intersection,sheet_index2]=intersection_arc_int_check(arcs,num_of_arcs,
      limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_i
      ndex,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_min,limit_in_max,limit_in_min,num_of_prods,flag_inde
      x,arc_a,arc_b,sheet_index2,arc_inner_no_int,arc_inner_no_int_index,arc_int_check,arc_outer_loop_index,arc_int_outer_loop_i
      ndex,arc_int_outer_sheet_index);
global flag,global arc,global temp5,global arc_int,global flag_no_intersection,global arcs_in, global arcs_out
flag_no_intersection(arc_inner_no_int(1,1),arc_inner_no_int(1,2)).sheet(arc_inner_no_int_index,arc_int_outer_loop_index)=1;
m=arc_int_outer_sheet_index;
n=1;
flag_overlap=0;
for k=1:num_of_prods
    %check to make sure all ranges of prods are ovelapping and intersection does exist
    if ((arc(arc_a,arc_b).limits(k,1,m)) <= (arc_int_check(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m))
     >= (arc_int_check(arc_a,arc_b).limits(k,1,n)))
      flag_overlap=flag_overlap+1;
   else
      if ((arc(arc_a,arc_b).limits(k,1,m)) >= (arc_int_check(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m))
        <= (arc_int_check(arc_a,arc_b).limits(k,1,n)))
         flag_overlap=flag_overlap+1;
      end
   end
end
if flag_overlap >= num_of_prods
   flag_no_intersection(arc_inner_no_int(1,1),arc_inner_no_int(1,2)).sheet(arc_inner_no_int_index,arc_int_outer_loop_index)=0;
end
if max(max(arc(arc_inner_no_int(1,1),arc_inner_no_int(1,2)).limits(:,:,arc_inner_no_int_index)))==0
   flag_no_intersection(arc_inner_no_int(1,1),arc_inner_no_int(1,2)).sheet(arc_inner_no_int_index,arc_int_outer_loop_index)=0;
end
```

## intersect_arc_star Function:

```
function[arc,arc_star,cleanup_flag,arc_int,flag_no_intersection,sheet_index2]=intersect_arc_star(arcs,num_of_arcs,limit_in_max
```

```
      _new,limit_in_min_new,limit_out_max_new,limit_out_min_new,k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_index,sheet_nu
      m_index,sheet_num_out_index,limit_out_max,limit_out_min,limit_in_max,limit_in_min,num_of_prods,flag_index,arc_a,arc_b,
      sheet_index2,flag_no_intersection)
global flag,global arc,global arc_star,global temp5, global arc_int,global flag_no_intersection,global sheet_index2,global arcs_in,
global arcs_out
empty_star=1;
empty_arc=1;
empty_special=0;
%intersection of non-special cases (ie. cases in which the min<>max for any product k
sheet_index=1;
temp5(arc_a,arc_b).limits=[];
temp6(arc_a,arc_b).limits=[];
arc_int(arc_a,arc_b).limits=[];
z_arc=(size(arc(arc_a,arc_b).limits,3));
z_star=(size(arc_star(arc_a,arc_b).limits,3));
sheet_index=1;
m=1;
while m<=z_arc
   n=1;
   flag_unique=0;
   cleanup_flag=[];
   while n<=z_star
      flag_updated=0;
      %this will find the intersection of two regions that share boundaries of num_of_prods-1 product(s)
      if flag_updated==0;
         flag_overlap=0;
         for k=1:num_of_prods
                  %check to make sure all ranges of prods are ovelapping and intersection does exist
            if ((arc(arc_a,arc_b).limits(k,1,m)) <= (arc_star(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m))
              >= (arc_star(arc_a,arc_b).limits(k,1,n)))
               flag_overlap=flag_overlap+1;
            else
               if ((arc(arc_a,arc_b).limits(k,1,m)) >= (arc_star(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m))
                 <= (arc_star(arc_a,arc_b).limits(k,1,n)))
                  flag_overlap=flag_overlap+1;
               end
            end
         end
         if flag_overlap >= num_of_prods%-1
            for k=1:num_of_prods
               temp5(arc_a,arc_b).limits(k,1,sheet_index)=max((arc(arc_a,arc_b).limits(k,1,m)),(arc_star(arc_a,arc_b).limits(k,1,n)));
               temp5(arc_a,arc_b).limits(k,2,sheet_index)=min((arc(arc_a,arc_b).limits(k,2,m)),(arc_star(arc_a,arc_b).limits(k,2,n)));
               flag_unique=1;
               flag_updated=1;
            end
         end
         %if flag_unique is equal to 0 if arc(m) has not been intersected with any other regions of arc_star(n)
         if flag_unique==0 &&  n==z_star
            temp5(arc_a,arc_b).limits(:,:,sheet_index)=arc(arc_a,arc_b).limits(:,:,m);
            flag_updated=1;
            arc_flag = 0;
         end
         if flag_updated==1
            sheet_index=sheet_index+1;
         end
      end
      n=n+1;
   end
   m=m+1;
end
z_temp=(size(temp5(arc_a,arc_b).limits,3));
j=1;
arc_star(arc_a,arc_b).limits=[];
arc_int(arc_a,arc_b).limits=temp5(arc_a,arc_b).limits;
```

## cleanup_sheets Function:

```
function[arc,arc_star]=cleanup_sheets(num_of_prods,arc_a,arc_b)
global arc,global arc_star,global flag,global U
tolerance = .25;
flag_equal=[];
a=arc_a;b=arc_b;
if size(arc(a,b).limits,2) > 1
   z=size(arc(a,b).limits,3);
   m=1;
   while m <= z
      for k=1:num_of_prods
         if (arc(a,b).limits(k,1,m)) > (arc(a,b).limits(k,2,m))
            flag_equal(k,1,m)=1;
         else
            flag_equal(k,1,m)=0;
         end
      end
      if max(max(flag_equal(:,:,m))) == 1
         arc(a,b).limits(:,:,m)=[];
         z=size(arc(a,b).limits,3);
         if m>=1 && m<=z
            m=m-1;
         end
      end
      m=m+1;
   end
end
flag_equal=[];
if size(arc(a,b).limits,1) > 2 && size(arc(a,b).limits,3) > 1
   z=size(arc(a,b).limits,3);
   for m=1:z
      if m <= z
         for k=1:num_of_prods
            if abs((arc(a,b).limits(k,1,m)) - (arc(a,b).limits(k,2,m)))<tolerance
               flag_equal(k,1,m)=1;
            else
               flag_equal(k,1,m)=0;
            end
         end
         if min(min(flag_equal(:,:,m))) == 1
            arc(a,b).limits(:,:,m)=[];
            z=size(arc(a,b).limits,3);
            if m>=1 && m<=z
               m=m-1;
            end
         end
      end
   end
end
%check if min and max limits are 0, if so delete sheet
if size(arc(a,b).limits,3) > 1
   z=size(arc(a,b).limits,3);
   m=1;
   while m<=z && m>0 && z>1
      j=1;
      while j<=num_of_prods
         if max(max(arc(a,b).limits(j,:,m)))==0
            arc(a,b).limits(:,:,m)=[];
            z=size(arc(a,b).limits,3);
            j=num_of_prods;
            m=0;
         end
         j=j+1;
      end
      m=m+1;
   end
end
%check if sheets are equal, if so delete sheet
if size(arc(a,b).limits,3) > 1
```

```
   z=size(arc(a,b).limits,3);
   m=1;
   while m<=z-1 && m>0
      flag_equal=[];
      n=m+1;
      while n<=z
         for k=1:num_of_prods
            if (size(arc(a,b).limits,3)>=m)&&(m>0)&&(n<=z)
               if abs( (arc(a,b).limits(k,1,m)) - (arc(a,b).limits(k,1,n))) < tolerance
                  flag_equal(k,1,n)=1;
               else
                  flag_equal(k,1,n)=0;
               end
               if abs( (arc(a,b).limits(k,2,m)) - (arc(a,b).limits(k,2,n))) < tolerance
                  flag_equal(k,2,n)=1;
               else
                  flag_equal(k,1,n)=0;
               end
            end
         end
         if min(min(flag_equal(:,:,n))) == 1
            for k=1:num_of_prods
               arc(a,b).limits(k,1,z+1)=min(arc(a,b).limits(k,1,m),arc(a,b).limits(k,1,n));
               arc(a,b).limits(k,2,z+1)=max(arc(a,b).limits(k,2,m),arc(a,b).limits(k,2,n));
            end
            arc(a,b).limits(:,:,m)=[];
            arc(a,b).limits(:,:,n-1)=[];
            z=size(arc(a,b).limits,3);
            n=z;
            m=m-1;
         end
         n=n+1;
      end
      m=m+1;
   end
end
```

## union_arc Function:

```
function[arc,cleanup_flag]=union_arc(arc_int,arcs,num_of_arcs,limit_in_max_new,limit_in_min_new,limit_out_max_new,
      limit_out_min_new,k,i,U,rho,l,node,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_ma
      x,limit_out_min,limit_in_max,limit_in_min,num_of_prods,flag_index,arc_a,arc_b)
global flag global arc
star=1;
empty_arc=1;
empty_special=0;
tolerance = .01;
tolerance_union = .01;
%the next few if statements are to remove subsets,supersets, and equivalent sheets from the arc matrix
z_arc=(size(arc(arc_a,arc_b).limits,3));
m=1;
sheet_index=1;
while m<=z_arc-1 && m>0
   n=1+m;
   flag_unique=0;
   cleanup_flag=[];
   while n<=z_arc
      flag_updated=0;
      if flag_updated==0;
         cleanup_flag(:,:,sheet_index)=[2];
         for k=1:num_of_prods
            if abs((arc(arc_a,arc_b).limits(k,1,m)) - (arc(arc_a,arc_b).limits(k,1,n))) < tolerance
               cleanup_flag(k,1,sheet_index)=0;
            else
               cleanup_flag(k,1,sheet_index)=2;
               break
            end
```

```
      if abs((arc(arc_a,arc_b).limits(k,2,m)) - (arc(arc_a,arc_b).limits(k,2,n))) < tolerance
         cleanup_flag(k,2,sheet_index)=0;
      else
         cleanup_flag(k,2,sheet_index)=2;
         break
      end
   end
   %if all are 0, then regions are equal
   if min(cleanup_flag(:,:,sheet_index))==0 & max(cleanup_flag(:,:,sheet_index))==0;
      arc(arc_a,arc_b).limits(:,:,n)=[];
      z_arc=(size(arc(arc_a,arc_b).limits,3));
      n=z_arc;
      m=0;
      flag_updated=1;
      flag_unique=1;
   end
end

if flag_updated==0
   cleanup_flag(:,:,sheet_index)=[2];
   for k=1:num_of_prods
      if (arc(arc_a,arc_b).limits(k,1,m)-tolerance <= arc(arc_a,arc_b).limits(k,1,n)) && (arc(arc_a,arc_b).limits(k,2,m)
        >= arc(arc_a,arc_b).limits(k,1,n)-tolerance)
         cleanup_flag(k,1,sheet_index)=-1;
      else
         cleanup_flag(k,1,sheet_index)=2;
         break
      end
      if  (arc(arc_a,arc_b).limits(k,2,m) >= arc(arc_a,arc_b).limits(k,2,n)-tolerance) && (arc(arc_a,arc_b).limits(k,1,m)-
       tolerance <= arc(arc_a,arc_b).limits(k,2,n))
         cleanup_flag(k,2,sheet_index)=-1;
      else
         cleanup_flag(k,2,sheet_index)=2;
         break
      end
   end
   %if all -1 then region arc(n) is complete subset of arc(m)
   if max(cleanup_flag(:,:,sheet_index))==-1 & min(cleanup_flag(:,:,sheet_index))==-1;
      arc(arc_a,arc_b).limits(:,:,n)=[];
      z_arc=(size(arc(arc_a,arc_b).limits,3));
      n=z_arc;
      m=0;%m-1;
      flag_updated=1;
      flag_unique=1;
   end
end
if flag_updated==0
   cleanup_flag(:,:,sheet_index)=[2];
   for k=1:num_of_prods
      if (arc(arc_a,arc_b).limits(k,1,m) >= arc(arc_a,arc_b).limits(k,1,n)-tolerance) && (arc(arc_a,arc_b).limits(k,1,m)-
       tolerance <= arc(arc_a,arc_b).limits(k,2,n))
         cleanup_flag(k,1,sheet_index)=1;
      else
         cleanup_flag(k,1,sheet_index)=2;
         break
      end
      if (arc(arc_a,arc_b).limits(k,2,m)-tolerance <= arc(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m)
       >= arc(arc_a,arc_b).limits(k,1,n)-tolerance)
         cleanup_flag(k,2,sheet_index)=1;
      else
         cleanup_flag(k,2,sheet_index)=2;
         break
      end
   end
   if min(cleanup_flag(:,:,sheet_index))==1 & max(cleanup_flag(:,:,sheet_index))==1;
      arc(arc_a,arc_b).limits(:,:,m)=[];
      z_arc=(size(arc(arc_a,arc_b).limits,3));
      n=z_arc;
      m=0;
      flag_updated=1;
```

```
            flag_unique=1;
         end
      end
      if flag_updated==1
         sheet_index=sheet_index+1;
      end
      n=n+1;
   end
   m=m+1;
end
%this will find the union of multiple regions that share boundaries of num_of_prods-1 product(s)
temp5(arc_a,arc_b).limits=[];
temp6(arc_a,arc_b).limits=[];
z_arc=(size(arc(arc_a,arc_b).limits,3));
m=1;
sheet_index=1;
flag_updated=0;
flag_updated2=0;
while m<=z_arc-1 && m>0
   n=1+m;
   flag_unique=0;
   while n<=z_arc
      equal_flag=0;
      for k=1:num_of_prods
         if abs(arc(arc_a,arc_b).limits(k,1,m) - arc(arc_a,arc_b).limits(k,1,n)) <= tolerance_union &&
           abs(arc(arc_a,arc_b).limits(k,2,m) - arc(arc_a,arc_b).limits(k,2,n)) <= tolerance_union
            equal_flag=equal_flag+1;
         end
      end
      if equal_flag >= num_of_prods-1
         flag_overlap=0;
         for k=1:num_of_prods
            if ((arc(arc_a,arc_b).limits(k,1,m))-tolerance_union <= (arc(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m))
                        >= (arc(arc_a,arc_b).limits(k,1,n))-tolerance_union)
               flag_overlap=flag_overlap+1;
            else
               if ((arc(arc_a,arc_b).limits(k,1,m)) >= (arc(arc_a,arc_b).limits(k,2,n))-tolerance_union &&
                           (arc(arc_a,arc_b).limits(k,2,m))-tolerance_union <= (arc(arc_a,arc_b).limits(k,1,n)))
                  flag_overlap=flag_overlap+1;
               end
            end
         end
         if flag_overlap == num_of_prods
            for k=1:num_of_prods
               temp6(arc_a,arc_b).limits(k,1)=min(arc(arc_a,arc_b).limits(k,1,m),arc(arc_a,arc_b).limits(k,1,n));
               temp6(arc_a,arc_b).limits(k,2)=max(arc(arc_a,arc_b).limits(k,2,m),arc(arc_a,arc_b).limits(k,2,n));
            end
            merged_sheet1=m;
            merged_sheet2=n;
            arc(arc_a,arc_b).limits(:,:,merged_sheet1)=[];
            arc(arc_a,arc_b).limits(:,:,merged_sheet2-1)=[];
            z_arc=(size(arc(arc_a,arc_b).limits,3));
            arc(arc_a,arc_b).limits(:,:,z_arc+1)=temp6(arc_a,arc_b).limits;
            n=z_arc;
            m=0;
            temp6(arc_a,arc_b).limits=[];
         end
      end
      n=n+1;
   end
   m=m+1;
end
%the next few if statements are to remove subsets,supersets, and equivalent sheets from the arc matrix
z_arc=(size(arc(arc_a,arc_b).limits,3));
m=1;
sheet_index=1;
while m<=z_arc-1 && m>0
   n=1+m;
   flag_unique=0;
   cleanup_flag=[];
```

```
while n<=z_arc
   flag_updated=0;
   if flag_updated==0;
      cleanup_flag(:,:,sheet_index)=[2];
      for k=1:num_of_prods
         if abs((arc(arc_a,arc_b).limits(k,1,m)) - (arc(arc_a,arc_b).limits(k,1,n))) <= tolerance
            cleanup_flag(k,1,sheet_index)=0;
         else
            cleanup_flag(k,1,sheet_index)=2;
            break
         end
         if abs((arc(arc_a,arc_b).limits(k,2,m)) - (arc(arc_a,arc_b).limits(k,2,n))) < tolerance
            cleanup_flag(k,2,sheet_index)=0;
         else
            cleanup_flag(k,2,sheet_index)=2;
            break
         end
      end
      %if all are 0, then regions are equal
      if min(cleanup_flag(:,:,sheet_index))==0 & max(cleanup_flag(:,:,sheet_index))==0;
         arc(arc_a,arc_b).limits(:,:,n)=[];
         z_arc=(size(arc(arc_a,arc_b).limits,3));
         n=z_arc;
         m=0;
         flag_updated=1;
         flag_unique=1;
      end
   end
   if flag_updated==0
      cleanup_flag(:,:,sheet_index)=[2];
      for k=1:num_of_prods
         if (arc(arc_a,arc_b).limits(k,1,m)-tolerance <= arc(arc_a,arc_b).limits(k,1,n)) && (arc(arc_a,arc_b).limits(k,2,m) >=
           arc(arc_a,arc_b).limits(k,1,n)-tolerance)
            cleanup_flag(k,1,sheet_index)=-1;
         else
            cleanup_flag(k,1,sheet_index)=2;
            break
         end
         if  (arc(arc_a,arc_b).limits(k,2,m) >= arc(arc_a,arc_b).limits(k,2,n)-tolerance) && (arc(arc_a,arc_b).limits(k,1,m)-tolerance
           <= arc(arc_a,arc_b).limits(k,2,n))
            cleanup_flag(k,2,sheet_index)=-1;
         else
            cleanup_flag(k,2,sheet_index)=2;
            break
         end
      end
      %if all -1 then region a1c(n) is complete subset of arc(m)
      if max(cleanup_flag(:,:,sheet_index))==-1 & min(cleanup_flag(:,:,sheet_index))==-1;
         arc(arc_a,arc_b).limits(:,:,n)=[];
         z_arc=(size(arc(arc_a,arc_b).limits,3));
         n=z_arc;
         m=0;
         flag_updated=1;
         flag_unique=1;
      end
   end
   if flag_updated==0
      cleanup_flag(:,:,sheet_index)=[2];
      for k=1:num_of_prods
         if (arc(arc_a,arc_b).limits(k,1,m) >= arc(arc_a,arc_b).limits(k,1,n)-tolerance) && (arc(arc_a,arc_b).limits(k,1,m)-tolerance
           <= arc(arc_a,arc_b).limits(k,2,n))
            cleanup_flag(k,1,sheet_index)=1;
         else
            cleanup_flag(k,1,sheet_index)=2;
            break
         end
         if (arc(arc_a,arc_b).limits(k,2,m)-tolerance <= arc(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m) >=
                     arc(arc_a,arc_b).limits(k,1,n)-tolerance)
            cleanup_flag(k,2,sheet_index)=1;
         else
```

```
            cleanup_flag(k,2,sheet_index)=2;
            break
          end
        end
        if min(cleanup_flag(:,:,sheet_index))==1 & max(cleanup_flag(:,:,sheet_index))==1;
          arc(arc_a,arc_b).limits(:,:,m)=[];
          z_arc=(size(arc(arc_a,arc_b).limits,3));
          n=z_arc;
          m=0;
          flag_updated=1;
          flag_unique=1;
        end
      end
      if flag_updated==1
        sheet_index=sheet_index+1;
      end
      n=n+1;
    end
    m=m+1;
end
%this will find the union of multiple regions that share boundaries of num_of_prods-1 product(s)
temp5(arc_a,arc_b).limits=[];
temp6(arc_a,arc_b).limits=[];
z_arc=(size(arc(arc_a,arc_b).limits,3));
m=1;
sheet_index=1;
flag_updated=0;
flag_updated2=0;
while m<=z_arc-1 && m>0
    n=1+m;
    flag_unique=0;
    while n<=z_arc
      equal_flag=0;
      for k=1:num_of_prods
        if abs(arc(arc_a,arc_b).limits(k,1,m) - arc(arc_a,arc_b).limits(k,1,n)) <= tolerance_union &&
              abs(arc(arc_a,arc_b).limits(k,2,m) - arc(arc_a,arc_b).limits(k,2,n)) <= tolerance_union
          equal_flag=equal_flag+1;
        end
      end
      if equal_flag >= num_of_prods-1
        flag_overlap=0;
        for k=1:num_of_prods
          if ((arc(arc_a,arc_b).limits(k,1,m))-tolerance_union <= (arc(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m))
                    >= (arc(arc_a,arc_b).limits(k,1,n))-tolerance_union)
            %check to make sure all ranges of prods are ovelapping and intersection does exist
            flag_overlap=flag_overlap+1;
          else
            if ((arc(arc_a,arc_b).limits(k,1,m)) >= (arc(arc_a,arc_b).limits(k,2,n))-tolerance_union &&
                      (arc(arc_a,arc_b).limits(k,2,m))-tolerance_union <= (arc(arc_a,arc_b).limits(k,1,n)))
              flag_overlap=flag_overlap+1;
            end
          end
        end
        if flag_overlap == num_of_prods
          for k=1:num_of_prods
            temp6(arc_a,arc_b).limits(k,1)=min(arc(arc_a,arc_b).limits(k,1,m),arc(arc_a,arc_b).limits(k,1,n));
            temp6(arc_a,arc_b).limits(k,2)=max(arc(arc_a,arc_b).limits(k,2,m),arc(arc_a,arc_b).limits(k,2,n));
          end
          merged_sheet1=m;
          merged_sheet2=n;
          arc(arc_a,arc_b).limits(:,:,merged_sheet1)=[];
          arc(arc_a,arc_b).limits(:,:,merged_sheet2-1)=[];
          z_arc=(size(arc(arc_a,arc_b).limits,3));
          arc(arc_a,arc_b).limits(:,:,z_arc+1)=temp6(arc_a,arc_b).limits;
          n=z_arc;
          m=0;
          temp6(arc_a,arc_b).limits=[];
        end
      end
      n=n+1;
```

```
   end
   m=m+1;
end
%the next few if statements are to remove subsets, supersets, and equivalent sheets from the arc matrix
z_arc=(size(arc(arc_a,arc_b).limits,3));
m=1;
sheet_index=1;
while m<=z_arc-1 && m>0
   n=1+m;
   flag_unique=0;
   cleanup_flag=[];
   while n<=z_arc
      flag_updated=0;
      if flag_updated==0;
         cleanup_flag(:,:,sheet_index)=[2];
         for k=1:num_of_prods
            if abs((arc(arc_a,arc_b).limits(k,1,m)) - (arc(arc_a,arc_b).limits(k,1,n))) <= tolerance
               cleanup_flag(k,1,sheet_index)=0;
            else
               cleanup_flag(k,1,sheet_index)=2;
               break
            end
            if abs((arc(arc_a,arc_b).limits(k,2,m)) - (arc(arc_a,arc_b).limits(k,2,n))) < tolerance
               cleanup_flag(k,2,sheet_index)=0;
            else
               cleanup_flag(k,2,sheet_index)=2;
               break
            end
         end
         %if all are 0, then regions are equal
         if min(cleanup_flag(:,:,sheet_index))==0 & max(cleanup_flag(:,:,sheet_index))==0;
            arc(arc_a,arc_b).limits(:,:,n)=[];
            z_arc=(size(arc(arc_a,arc_b).limits,3));
            n=z_arc;
            m=0;
            flag_updated=1;
            flag_unique=1;
         end
      end
      if flag_updated==0
         cleanup_flag(:,:,sheet_index)=[2];
         for k=1:num_of_prods
            if (arc(arc_a,arc_b).limits(k,1,m)-tolerance <= arc(arc_a,arc_b).limits(k,1,n)) && (arc(arc_a,arc_b).limits(k,2,m) >=
                        arc(arc_a,arc_b).limits(k,1,n)-tolerance)
               cleanup_flag(k,1,sheet_index)=-1;
            else
               cleanup_flag(k,1,sheet_index)=2;
               break
            end
            if  (arc(arc_a,arc_b).limits(k,2,m) >= arc(arc_a,arc_b).limits(k,2,n)-tolerance) && (arc(arc_a,arc_b).limits(k,1,m)-tolerance
                        <= arc(arc_a,arc_b).limits(k,2,n))
               cleanup_flag(k,2,sheet_index)=-1;
            else
               cleanup_flag(k,2,sheet_index)=2;
               break
            end
         end
         %if all -1 then region a1c(n) is complete subset of arc(m)
         if max(cleanup_flag(:,:,sheet_index))==-1 & min(cleanup_flag(:,:,sheet_index))==-1;
            arc(arc_a,arc_b).limits(:,:,n)=[];
            z_arc=(size(arc(arc_a,arc_b).limits,3));
            n=z_arc;
            m=0;
            flag_updated=1;
            flag_unique=1;
         end
      end
      if flag_updated==0
         cleanup_flag(:,:,sheet_index)=[2];
         for k=1:num_of_prods
```

```
            if (arc(arc_a,arc_b).limits(k,1,m) >= arc(arc_a,arc_b).limits(k,1,n)-tolerance) && (arc(arc_a,arc_b).limits(k,1,m)-tolerance
                        <= arc(arc_a,arc_b).limits(k,2,n))
               cleanup_flag(k,1,sheet_index)=1;
            else
               cleanup_flag(k,1,sheet_index)=2;
               break
            end
            if (arc(arc_a,arc_b).limits(k,2,m)-tolerance <= arc(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m) >=
                        arc(arc_a,arc_b).limits(k,1,n)-tolerance)
               cleanup_flag(k,2,sheet_index)=1;
            else
               cleanup_flag(k,2,sheet_index)=2;
               break
            end
         end
         if min(cleanup_flag(:,:,sheet_index))==1 & max(cleanup_flag(:,:,sheet_index))==1;
            arc(arc_a,arc_b).limits(:,:,m)=[];
            z_arc=(size(arc(arc_a,arc_b).limits,3));
            n=z_arc;
            m=0;
            flag_updated=1;
            flag_unique=1;
         end
      end
      if flag_updated==1
         sheet_index=sheet_index+1;
      end
      n=n+1;
   end
   m=m+1;
end
%this will find the union of multiple regions that share boundaries of num_of_prods-1 product(s)
temp5(arc_a,arc_b).limits=[];
temp6(arc_a,arc_b).limits=[];
z_arc=(size(arc(arc_a,arc_b).limits,3));
m=1;
sheet_index=1;
flag_updated=0;
flag_updated2=0;
while m<=z_arc-1 && m>0
   n=1+m;
   flag_unique=0;
   while n<=z_arc
      equal_flag=0;
      for k=1:num_of_prods
         if abs(arc(arc_a,arc_b).limits(k,1,m) - arc(arc_a,arc_b).limits(k,1,n)) <= tolerance_union &&
               abs(arc(arc_a,arc_b).limits(k,2,m) - arc(arc_a,arc_b).limits(k,2,n)) <= tolerance_union
            equal_flag=equal_flag+1;
         end
      end
      if equal_flag >= num_of_prods-1
         flag_overlap=0;
         for k=1:num_of_prods
            if ((arc(arc_a,arc_b).limits(k,1,m))-tolerance_union <= (arc(arc_a,arc_b).limits(k,2,n)) && (arc(arc_a,arc_b).limits(k,2,m))
                        >= (arc(arc_a,arc_b).limits(k,1,n))-tolerance_union)
               %check to make sure all ranges of prods are ovelapping and intersection does exist
               flag_overlap=flag_overlap+1;
            else
               if ((arc(arc_a,arc_b).limits(k,1,m)) >= (arc(arc_a,arc_b).limits(k,2,n))-tolerance_union &&
                           (arc(arc_a,arc_b).limits(k,2,m))-tolerance_union <= (arc(arc_a,arc_b).limits(k,1,n)))
                  flag_overlap=flag_overlap+1;
               end
            end
         end
         if flag_overlap == num_of_prods
            for k=1:num_of_prods
               temp6(arc_a,arc_b).limits(k,1)=min(arc(arc_a,arc_b).limits(k,1,m),arc(arc_a,arc_b).limits(k,1,n));
               temp6(arc_a,arc_b).limits(k,2)=max(arc(arc_a,arc_b).limits(k,2,m),arc(arc_a,arc_b).limits(k,2,n));
            end
            merged_sheet1=m;
```

```
            merged_sheet2=n;
            arc(arc_a,arc_b).limits(:,:,merged_sheet1)=[];
            arc(arc_a,arc_b).limits(:,:,merged_sheet2-1)=[];
            z_arc=(size(arc(arc_a,arc_b).limits,3));
            arc(arc_a,arc_b).limits(:,:,z_arc+1)=temp6(arc_a,arc_b).limits;
            n=z_arc;
            m=0;
            temp6(arc_a,arc_b).limits=[];
         end
      end
      n=n+1;
   end
   m=m+1;
end
```

## set_change_flag Function:

```
function[change_flag,arc_stored,arc_stored_index,arc_intersected]=set_change_flag(arc,arc_org,num_of_prods,arc_a,arc_b,
      flag_index);
global change_flag global arc_stored global arc_stored_index global arc_intersected, global arc_int
tolerance = .25;
z=size(arc(arc_a,arc_b).limits,3);
z_org=size(arc_org(arc_a,arc_b).limits,3);
temp_flag=0;
m=1;
if size(arc(arc_a,arc_b).limits,2)>0
   while m<=z && m>0
      flag_equal=[0];
      n=1;
      while n<=z_org
         for k=1:num_of_prods
            if abs((arc(arc_a,arc_b).limits(k,1,m)) - (arc_org(arc_a,arc_b).limits(k,1,n))) <= tolerance
               flag_equal(k,1,n)=1;
            else
               flag_equal(k,1,n)=0;
            end
            if abs((arc(arc_a,arc_b).limits(k,2,m)) - (arc_org(arc_a,arc_b).limits(k,2,n))) <= tolerance
               flag_equal(k,2,n)=1;
            else
               flag_equal(k,1,n)=0;
            end
         end
         if min(min(flag_equal(:,:,n))) == 1%the two regions are equal
            temp_flag=1+temp_flag;
            n=z_org;
            break
         end
         n=n+1;
      end
      if n == z_org+1 && min(min(min(flag_equal))) == 0
         arc(arc_a,arc_b).limits(:,:,m);
      end
      m=m+1;
   end
   arc_stored(arc_a,arc_b,flag_index,arc_stored_index).limits=arc(arc_a,arc_b).limits;
   arc_stored_index=1+arc_stored_index;
   if temp_flag < z_org & z > 0;
      change_flag = 1;
   end
end
```

## update_change_flag_out Function:

```
function[flag_length,flag]=update_change_flag_out(i,arc_in,arc_out,arcs_in,arcs_out)
global flag,global flag_length,global change_flag, global flag_ini
```

```
flag_length=length(flag);
flag_new=0;
if change_flag == 1 && flag_length > 0
   flag_new = flag;
   present_out=0;
   present_current=0;
   for index_flg_lth=1:flag_length %looks for the outgoing node in the flag
      if flag(index_flg_lth) == arc_out(1,2)
         present_out=1;
      end
      if flag(index_flg_lth) == i
         present_current=1;
      end
   end
   if present_current == 0%adds current node to flag if not present
      flag_new(flag_length+1)=i;
   end
   flag_length_new=length(flag_new);
   if present_out == 0%adds node out to flag if not present
      flag_new(flag_length_new+1)=arc_out(1,2);
   end
   flag=flag_new;
end
flag_length=length(flag);
flag_new=0;
if change_flag == 1 && flag_length == 0
   flag_new(2)=i;
   flag_new(3)=arc_out(1,2);
   flag=flag_new;
end
flag_length=length(flag);
change_flag=0;
```

## update_change_flag Function:

```
function[flag_length,flag]=update_change_flag(i,arc_in,arc_out,arcs_in,arcs_out)
global flag,global flag_length,global change_flag, global flag_ini
flag_length=length(flag);
flag_new=0;
if change_flag == 1 && flag_length > 0
   for arc_flag_in_index=1:size(arcs_in,1)
      flag_new = flag;
      present_in=0;
      present_current=0;
      for index_flg_lth=1:flag_length %looks for the outgoing node in the flag
         if flag(index_flg_lth) == arcs_in(arc_flag_in_index,1)
            present_in=1;
         end
         if flag(index_flg_lth) == i
            present_current=1;
         end
      end
      if present_current == 0%adds current node to flag if not present
         flag_new(flag_length+1)=i;
      end
      flag_length_new=length(flag_new);
      if present_in == 0%adds node out to flag if not present
         flag_new(flag_length_new+1)=arcs_in(arc_flag_in_index,1);
      end
      flag=flag_new;
   end
end
flag_length=length(flag);
flag_new=0;
if change_flag == 1 && flag_length == 0
   flag_new(2)=i;
   for arc_flag_in_index=1:size(arcs_in,1)
```

```
      flag_new(2+arc_flag_in_index)=arcs_in(arc_flag_in_index,1);
   end
   flag=flag_new;
end
change_flag=1;
flag_length=length(flag);
flag_new=0;
if change_flag == 1 && flag_length > 0
   for arc_flag_out_index=1:size(arcs_out,1)
      flag_new = flag;
      present_out=0;
      present_current=0;
      for index_flg_lth=1:flag_length %looks for the outgoing node in the flag
         if flag(index_flg_lth) == arcs_out(arc_flag_out_index,2);
            present_out=1;
         end
         if flag(index_flg_lth) == i
            present_current=1;
         end
      end
      if present_current == 0%adds current node to flag if not present
         flag_new(flag_length+1)=i;
      end
      flag_length_new=length(flag_new);
      if present_out == 0%adds node out to flag if not present
         flag_new(flag_length_new+1)=arcs_out(arc_flag_out_index,2);
      end
      flag=flag_new;
   end
   flag_length=length(flag);
   flag_new=0;
   if change_flag == 1 && flag_length == 0
      flag_new(2)=i;
      for arc_flag_out_index=1:size(arcs_out,1)
         flag_new(2+arc_flag_out_index)=arcs_out(arc_flag_out_index,2);
      end
      flag=flag_new;
   end
end
flag_length=length(flag);
change_flag=0;
```

## calculate_new_in_limits Function:

```
function[limit_in_max_new,limit_in_min_new]=calculate_new_in_limits(k,i,U,rho,l,node,arc,arc_in,arc_out,arc_inner_loop_index,sh
eet_num_index,sheet_num_out_index,limit_out_max,limit_out_min,limit_in_max,limit_in_min,idle_flag)
global limit_in_max_new,global limit_in_min_new
if node(i).type == 1 %for idle node
   if node(arc_out(1,2)).product_num ~= k
      idle_time_min=max(0,node(i).time_delta_min);
      idle_time_max=max(0,node(i).time_delta_max);
      limit_in_min_new(k)=min(limit_out_min(k)+idle_time_min*rho(k),limit_out_max(k)+idle_time_min*rho(k));
      limit_in_max_new(k)=max(limit_out_max(k)+idle_time_max*rho(k),limit_out_min(k)+idle_time_max*rho(k));
   end
end
if node(i).type == 2 %for setup node
   limit_in_min_new(k)=limit_out_min(k) + node(i).time_delta*rho(k) ;
   limit_in_max_new(k)=limit_out_max(k) + node(i).time_delta*rho(k) ;
end
if node(i).type ~= 3 %for fill node
   if node(i).product_num == k
      limit_in_min_new(k)=limit_out_min(k) + node(i).time_delta_max*rho(k);
      limit_in_max_new(k)=limit_out_max(k) + node(i).time_delta_min*rho(k);
   end
end
```

## calculate_y_star Function:

```
function[arc_star,change_flag]=calculate_y_star(limit_in_max_new,limit_in_min_new,limit_out_max_new,limit_out_min_new,k,i,U,
      rho,l,node,arc,arc_in,arc_out,arc_inner_loop_index,sheet_num_index,sheet_num_out_index,limit_out_max,limit_out_min,limit_
      in_max,limit_in_min,num_of_prods,num_of_arcs,arcs);
global arc_star, global change_flag
for o=1:num_of_prods
   arc_star(arc_in(1),arc_in(2)).limits(o,:,arc_inner_loop_index)=[0 0;];
end
for k=1:num_of_prods
   arc_star(arc_in(1,1),arc_in(1,2)).limits(k,2,arc_inner_loop_index)=limit_in_max_new(k);
   arc_star(arc_in(1,1),arc_in(1,2)).limits(k,1,arc_inner_loop_index)=limit_in_min_new(k);
end
flag_equal=[];
a=arc_in(1,1);b=arc_in(1,2);
if size(arc_star(a,b).limits,3) >= 1
   z=size(arc_star(a,b).limits,3);
   m=arc_inner_loop_index;
   while m <= z
      for k=1:num_of_prods
         if (arc_star(a,b).limits(k,1,m)) == (arc_star(a,b).limits(k,2,m))
            flag_equal(k,1)=1;
         else
            flag_equal(k,1)=0;
         end
      end
      if min(min(flag_equal(:,:))) == 1
         arc_star(a,b).limits(:,:,m)=0;
         z=size(arc_star(a,b).limits,3);
      end
      flag_equal=[];
      m=m+1;
   end
end
flag_equal=[];
a=arc_in(1,1);b=arc_in(1,2);
if size(arc_star(a,b).limits,3) >= 1
   z=size(arc_star(a,b).limits,3);
   m=arc_inner_loop_index;
   while m <= z
      for k=1:num_of_prods
         if (arc_star(a,b).limits(k,1,m)) > (arc_star(a,b).limits(k,2,m))
            flag_equal(k,1)=1;
         else
            flag_equal(k,1)=0;
         end
      end
      if max(min(flag_equal(:,:))) == 1
         arc_star(a,b).limits(:,:,m)=0;
         z=size(arc_star(a,b).limits,3);
      end
      flag_equal=[];
      m=m+1;
   end
end
```

## update_change_flag_in Function:

```
function[flag_length,flag]=update_change_flag_in(i,arc_in,arc_out,arcs_in,arcs_out)
global flag,global flag_length,global change_flag, global flag_ini
flag_length=length(flag);
flag_new=0;
if change_flag == 1 && flag_length > 0
   flag_new = flag;
   present_in=0;
   present_current=0;
   for index_flg_lth=1:flag_length %looks for the outgoing node in the flag
      if flag(index_flg_lth) == arc_in(1,1)
```

```
        present_in=1;
      end
      if flag(index_flg_lth) == i
        present_current=1;
      end
    end
    if present_current == 0%adds current node to flag if not present
      flag_new(flag_length+1)=i;
    end
    flag_length_new=length(flag_new);
    if present_in == 0%adds node out to flag if not present
      flag_new(flag_length_new+1)=arc_in(1,1);
    end
    flag=flag_new;
end
flag_length=length(flag);
flag_new=0;
if change_flag == 1 && flag_length == 0
    flag_new(2)=i;
    flag_new(3)=arc_in(1,1);
    flag=flag_new;
end
flag_length=length(flag);
change_flag=0;
```

## update_node_number Function:

```
function[flag_length,flag]=update_node_number(i)
global flag,global flag_length,global change_flag
flag_length=length(flag);
flag_new=0;
if (flag_length>1) && (flag(2)>0)
    for j=2:flag_length
      flag_new(j-1) = flag(j);
    end
end
if (flag_length>1) && (flag(2)== 0)
    for j=3:flag_length
      flag_new(j-2) = flag(j);
    end
end
if (flag_length == 1 || flag_length == 0) && (change_flag == 0)
    flag_new = '';
end
flag=flag_new;
flag_length=length(flag);
```

## Appendix II: Sequencing Algorithm Implementation

**Algorithm**

The sequencing algorithm functions by evaluating a set of products that are at or below the buffer threshold. The algorithm calculates a goodness value for each product based on a weighted goodness equation for the system and the product with the highest goodness value is selected as the next product to be sequenced. The algorithm also allows the user to define a lookahead time window to calculate future goodness values for each future sequence step within the window. The first product of the highest valued sequence is selected to be the next product to be sequenced. The lookahead function allows the algorithm to predict future states in order to avoid dead-end paths.

**Description Implemented Sequencing Algorithm**

This section provides a description of how the implemented sequencing algorithm functions. Note that the definition of key variables can be found in Section 6.4.1 Key Variables.

Functions:

- initialization(): function to define various parameters of a given production system. The parameters defined by the function include the setup times, production rates, usage rates, maximum buffer levels and thresholds and initial buffer levels.
- threshold_check($BF_i(t)$): function that cycles through all products to find products that have reached or crossed the $BF\_threshold_i$. Flagged products are then stored in the *prod_selection* list.
- coefficient_normalization($BF_i(t)$,$PR_i$,$UR_i$,$cost(i,j)$): function that normalizes the weighting factors $\alpha$, $\beta$, $\gamma$, $\varepsilon$, and $\eta$ such that each term of the goodness equation corresponds to the initial weighting factor value and the sum of the equation is one.
- current_state_selection($BF_i(t)$,$PR_i$,$UR_i$,$cost(i,j)$,$\alpha$,$\beta$,$\gamma$,$\varepsilon$,$\eta$): this function calculates the goodness value for all products contained in the *prod_selection* list. The function returns the *prod_selection* list ranked from the best product to worst product.
- lookahead_selection($BF_i(t)$,$PR_i$,$UR_i$,$cost(i,j)$,$\chi$,$\psi$,$\varphi$,$\omega$,$\zeta$,*la_time*): this function calculates the lookahead goodness value for *la_time* into the future, the first products considered are contained in the *prod_selection* list, but any product that reaches the $BF\_threshold_i$ is considered for subsequent sequence steps. The function returns the best first product from all the sequences considered within the lookahead window.

The initialization function defines the starting buffer levels and maximum buffer levels, production and usage rates, changeover costs, buffer thresholds, etc. The initialization function also allows the user to define the weighting factors for the current_state_selection function.

The algorithm will then call the *threshold_check* function to evaluate if any products are below the buffer threshold, $BF\_threshold_i$. If no products are below the threshold then the system is idle until a product crosses the threshold. The amount of idle time is calculated by advancing a user defined, Δ, number of time units into the future. The buffer levels are recalculated after each time step for all products, by subtracting the number of products consumed ($\text{consumed}(i) = \Delta \times UR(i)$) from the previous buffer level. The buffer threshold is checked after each Δ time units. This continues until one or more products are below the buffer threshold level, $BF\_threshold_i$.

The product or products are flagged in the *prod_selection* variable when one or more products cross the buffer threshold whether or not idle time was experienced by the system. The time at which the threshold was crossed is recorded for each flagged product. The *threshold_check* function is then exited.

The coefficient_normalization function is then called to determine the normalized the weighting factors for the goodness equations to increase the effectiveness of each factor. The normalizing coefficient variable is determined first by calculating the maximum value of the term that the coefficient will be applied to (i.e. time to crash, time to refill, time in queue, changeover cost, or Miltenburg's usage rate variation). The initial value of the weighting factor is then divided by the corresponding normalizing coefficient variable and the new normalized weighting factor value is used for the goodness calculations.

The *current_state_selection* function begins by considering the products contained in the *prod_selection* list. If only one product is contained in the *prod_selection* variable, the *current_state_selection* function is exited. When multiple products are below the buffer threshold, the goodness equation is used to evaluate the products.

All products in the *prod_selection* variable are evaluated and the goodness values are stored. The products are then sorted and ranked and the product with the highest goodness value is selected to be the next product in the sequence.

If the lookahead time is set to zero, *lookahead_selection* function is skipped. If the lookahead time is greater than zero, the *lookahead_selection* function is entered. The *lookahead_selection* function will project all possible production sequences the defined amount of lookahead time into the future. Each sequence is evaluated based on a lookahead goodness equation.

After evaluating all the sequences for the defined amount of lookahead time, the goodness values are stored and the *lookahead_selection* function is exited. The sequences are sorted and ranked based upon the goodness values and the first product of the highest valued sequence is selected as the next product to be refilled.

If only one product is to be sequenced, the algorithm stops. If multiple products are to be sequenced by the algorithm, the refill time is calculated for the selected product, whether selected by the *current_state_selection* function or the *lookahead_selection* function. All other product buffers are then decreased based upon the usage rate of the given product and the refill time of the selected product. The selected product is refilled and the product number is recorded. Then the time is advanced by the refill time and the process is repeated the desired number of times.

**Implementation of Algorithm**

This algorithm was implemented using MATLAB Release 14, Version 7.0.4.

**Complexity of Algorithm**

The computational complexity of the algorithm has not been calculated formally but the author acknowledges that the algorithm is not optimized. The algorithm was developed based on the overall function of the code instead of optimization of the computational time.

An example of the function of the code taking precedence over the computational efficiency is that Miltenburg's usage rate variation is calculated multiple times at different points in the algorithm. A more efficient approach could be to store each iteration of the calculation so that only the required portion must be recalculated instead starting at the beginning each time.

Note that the combination of high buffer threshold levels, large number of products, and a large lookahead time will significantly increase computation time. The number of products to be evaluated at each step potentially increases as the buffer threshold parameter increases. Consider that a 100% buffer threshold value will cause every product to be evaluated at every step. Also if a large lookahead time is defined then all products will be propagated the defined large amount of time into the future to evaluate all possible sequences at each step of the algorithm, which could potentially be thousands of sequences. Therefore some caution should be used when defining the lookahead time and buffer threshold values when there are a large number of products in the system.

**Sequencing Algorithm Source Code:**

```
clear all;
global alpha,global beta,global gamma_,global eta,global eta_val,global x_,global y_, global num_of_prod global BF_init,global
BF_maxt,global BF_tcross,global percentage,global BF_ini global idle_time,global time_increment,global marker,global
record,global time_stamp global step,global ans1_size,global prod_thrs,global cost_threshold global lookahead_time,global
BF_threshold,global delta,global epsilon,global chi,global psi,global omega,global time_stamp global marker,global cost,global
PR,global BF_ini,global BF_max,global UR, global step,global BF_store,global selection_num, global prod_thrs,global record,global
alphap, global betap, global gamma_p;global epsilon_p, global etap,global prod_selection,global ans2_size,global IX,global
cur_state,global alpha_record,global beta_record,global gamma_record, global epsilon_record,global eta_record,global seq_goodness,
global goodness_tr,global goodness_tr_time,global pg_num,global milt_UR,global IX,global num_of_prod_sequenced
global zeta_,global omega, global phi,global goodness_flag

previous_prod=1;
num_of_prod=8;
initialize(previous_prod,num_of_prod);
time_to_crash(1,1)=0;
time_to_refill(1,1)=0;
bad=0;
alpha =     .2 ;% %bf/ur coefficient - it's the time until buffer is empty
beta =      .2 ;% %(BF_max-bf)/pr coefficient - it's the time to refill the buffer
gamma_ =    .2 ;%%(t-tr) coefficient - the time since buffer was refilled
epsilon =   .2 ;%%c/o cost coefficient - the time to change from previous product to current product
eta_val =   .2 ;%milt's weighting factor
alpha_val = alpha;
beta_val = beta;
gamma_val = gamma_;
epsilon_val = epsilon;%
max_steps=200;
for step=1:max_steps; %to repeat program for max_steps, to generate multiple step sequence for debugging
   final_sequence(step,1)=0;
   record(step+1,num_of_prod+1)=0.;%set starting time to zero
   threshold_check(previous_prod,num_of_prod,BF_max,BF_threshold,delta,UR,cost,cost_threshold);
   coef_norm(BF_ini,BF_max,UR,PR,BF_threshold);
   alpha = alphap*alpha_val;
   beta = betap * beta_val; %
   gamma_ = gamma_p * gamma_val;
   epsilon = epsilon_p * epsilon_val;
   if idle_time == 0
      cur_state_selection_fnx(num_of_prod,step,record,time_stamp,BF_max,BF_ini,UR,PR,ans1_size,prod_thrs,selection_num,
        previous_prod,final_sequence);
      first=prod_selection(1);
      done=0;
      if lookahead_time > 0 && sum(prod_thrs) > 1
         test_combined_sequence(previous_prod,num_of_prod,lookahead_time,UR,PR,cost,cost_threshold,BF_threshold,
              BF_ini,BF_max,BF_tcross,prod_thrs,prod_selection,ans2_size,chi,psi);
         global seq_goodness,global goodness_tr,global goodness_tr_time,global goodness_BF_level,global seq_goodness_stored
         if pg_num ==2
            seq_goodness(:,:,2)=seq_goodness(:,:,1);
         end
```

```
        if max(max(seq_goodness(:,:,size(seq_goodness,3)))) > 0
           goodness_sheet=size(seq_goodness,3);
        elseif max(max(seq_goodness(:,:,size(seq_goodness,3)-1))) > 0
           goodness_sheet=size(seq_goodness,3)-1;
        end
        A=[];
        A = sortrows(seq_goodness_stored(:,:,goodness_sheet),[-(goodness_sheet+1) -goodness_sheet]);%this selects max(min of
                                                                                                    sequence goodness)
        %A = sortrows(seq_goodness_stored(:,:,goodness_sheet),[-goodness_sheet]);%this selects highest average goodness value
         A_stored(1:size(A,1),1:size(A,2),step)=A;
         for rank_index=1:size(A,1)
            if A(rank_index,goodness_sheet+3)~= 0
               if seq_goodness(A(rank_index,goodness_sheet+3),4,goodness_sheet) ~= 0
                  first=seq_goodness(A(rank_index,goodness_sheet+3),4,goodness_sheet);
                  break
               end
            end
         end
      end
      if first == 0
         break
      else
      refill_time=[BF_max(first)-BF_ini(first)+UR(first)*cost(previous_prod,first)]/(PR(first)
        -UR(first))+cost(previous_prod,first);%calc time to refill product chosen by lookahead module
      for i = 1:num_of_prod; %cycle thru products and subtracted used quantities
        if i == first; %updates the BF_ini for the product being replenished, or other products not replenished
           BF_ini(first)=BF_max(first); %update buffer qty
        else
           used = UR(i) * refill_time;
           BF_ini(i)=BF_ini(i) - used; %update buffer level
        end
      end
      record (step+1,num_of_prod+3)= first;
      time_to_crash (step+1,num_of_prod+3)= first;
      time_to_refill(step+1,num_of_prod+3)= first;
      final_sequence(step,1)=first;
      record(step+1,num_of_prod+6)=cost(previous_prod,first);
      previous_prod=first;
      record (step+1,1) = refill_time+record(step,1);
      marker(final_sequence(step,1)) = 0;%to update the marker to record the time when the product is refilled
      time_stamp(final_sequence(step,1)) = 0;%to reset the time at which the threshold is crossed

      for i=1:num_of_prod;
        record (step+1,i+1)=BF_ini(i)/BF_max(i);%percent fullness of buffer
        if record (step+1,i+1) < 0
           bad = 100;
        end
      end
      time_to_crash (step+1,1) = record(step,1);
      time_to_refill (step+1,1) = record(step,1);
      for i=1:num_of_prod;
        time_to_crash (step+1,i+1)=BF_ini(i)/UR(i);%time to crash
        time_to_refill (step+1,i+1)=(BF_max(i)-BF_ini(i))/(PR(i)-UR(i));%time to refill
      end
      prod_selection=[];
   else
      time_to_crash (step+1,1) = record(step,1);
      time_to_refill (step+1,1) = record(step,1);
      for i=1:num_of_prod;
        time_to_crash (step+1,i+1)=BF_ini(i)/UR(i);%time to crash
        time_to_refill (step+1,i+1)=(BF_max(i)-BF_ini(i))/(PR(i)-UR(i));%time to refill
      end
      step;
      prod_selection=[];
   end %this is the end of the loop to skip the lookahead if prod_selection only has one product
   if bad ~= 0
      break
   end
   loop_step=1;
end %this is the end on the loop to step thru the algorithm
```

## initialize Function:

```
function [cost_threshold,lookahead_time,BF_threshold,delta,alpha,beta,gamma_,epsilon,eta,chi,psi,omega,marker,cost,
BF_ini,BF_max,PR,UR,selection_num,prod_threshold,alphap,betap,gamma_p,epsilon_p,etap]=initialize(first_prod,num_of_prod)
global cost_threshold,global lookahead_time,global BF_threshold,global delta,global alpha,global beta,global gamma_,global epsilon;
global chi,global psi,global omega,global time_stamp,global marker,global cost,global PR,global BF_ini,global BF_max;
global UR,global selection_num;


cost_threshold= 8; %max cost that will be considered, all costs great will be considered infinite
cost =5*[1 1 2 1 2 1 2 1 1 1;%1
        1 1 1 2 1 1 1 1 1 1;%2
        1 2 1 1 2 1 1 1 1 1;%3
        1 1 2 1 1 2 2 1 1 1;%4
        1 2 1 1 1 2 2 1 1 1;%5
        1 1 2 1 2 1 1 1 1 1;%6
        1 1 1 1 1 1 1 1 1 1;%7
        1 1 1 1 1 1 1 1 1 1;%8
        1 1 1 1 1 1 1 1 1 1;%9
        1 1 1 1 1 1 1 1 1 1;];%10
PR=3*[5 5 5 5 5 .30 .300 .300 .300 .30;]; %rate of production, replenishment
UR=1*[4 3 2 2 1 .05 .05 .05 .05 .05;]; %usage/consumption rate (parts per time unit)
BF_ini =40*[100 100 100 100 100 275 275 275 275 298;]; %initial buffer, in num of prods
BF_max =40*[100 100 100 100 100 300 300 300 300 300;]; %max buffer levels in num of prods
BF_threshold=.95*[1 1 1 1 1   1 1 1 1 1   1 1 1 1 1   1 1 1 1 1   1 1 1 1 1   1 1 1 1 1;];%buffer fullness threshold, percentage of how full
the buffer is.  buffers that are lower will be considered
lookahead_time= 10.0; %# of time units to lookahead when finding a "best" sequence
delta=1;%the number of time units to advance when idle time occurs--how far to step ahead
selection_num=num_of_prod;%num of products to be selected by the goodness function to be tested by the lookahead function
time_stamp(num_of_prod) = [0];
marker(num_of_prod) = [0];
prod_thrs(num_of_prod) = [0];
alphap=1;
betap=1; %
gamma_p=1;
epsilon_p=1;
etap=1;
```

## threshold_check Function:

```
function [BF_ini,BF_max,BF_init,BF_maxt,BF_tcross,step,record,marker,time_stamp,ans1_size,prod_thrs,time_increment]=
   threshold_check(previous_prod,num_of_prod,BF_max,BF_threshold,delta,UR,cost,cost_threshold)
global BF_init,global BF_ini,global BF_maxt,global time_stamp,global marker,global idle_time,global step,global record,
global ans1_size,global prod_thrs,global time_increment
for i=1:num_of_prod;
   BF_init(i)=BF_ini(i)/UR(i); %convert into time units for the time until crash (at zero)
   BF_maxt(i)=BF_max(i)/UR(i); %convert into time units for max possible time until crash
   BF_tcross(i)=(BF_max(i)*BF_threshold(i)-BF_ini(i))/UR(i);%calc time until prod will cross bf threshold
end
% this will calculate the number of products that are below the buffer threshold, i.e. the products that need to be refilled
time_increment=0;
idle_time=0;
for increment=1:1000
   for i=1:num_of_prod;
      if (BF_ini(i)/BF_max(i) <= BF_threshold(i)) && (cost(previous_prod,i) <= cost_threshold)
         prod_thrs(i)=1;
      else
         prod_thrs(i)=0;
      end
   end
   ans1=find(prod_thrs == 1);
   ans1_size=size(ans1);
   if  (ans1_size(1,2) < 1);
      idle_time=1;
      time_increment=time_increment+delta;
      for j=1:num_of_prod;
```

```
        BF_ini(j) = BF_ini(j) - UR(j)*delta;
        BF_init(j)=BF_ini(j)/UR(j); %convert into time units
              BF_tcross(j)=(BF_max(j)*BF_threshold(j)-BF_ini(j))/UR(j);%calc time until prod reaches bf_thres
      end
   else
      break;
   end
end
% this will record any idle time if present and advance the record matrix appropriately
if idle_time==1;
   record(step+1,1)=record(step,1)+time_increment;
   record(step,num_of_prod+2)=idle_time;
   for i=1:num_of_prod
      record(step+1,i+1)=BF_ini(i)/BF_max(i);
   end
end
%this is to record the time when the buffer level drops below the buffer threshold, current time becomes the time_stamp(i)
for i=1:num_of_prod;
   if  ((BF_ini(i)/BF_max(i) <= BF_threshold(i)) && marker(i) == 0);
      if step < 2
         time_stamp(i) = -0.1;
         marker(i) = 1;
      else
         time_stamp(i) = record(step,1)-(BF_threshold(i)*BF_max(i)-BF_ini(i))/UR(i);
         marker(i) = 1;
      end
   end
end
```

## coef_norm Function:

```
function [alphap,betap,gamma_p,epsilon_p]=coef_norm(BF_ini,BF_max,UR,PR,BF_threshold)
global step,global record,global alphap, global betap, global gamma_p, global num_of_prod,global epsilon_p,global cost
for i = 1:num_of_prod
   bf_avg(i) = BF_ini(i)/BF_max(i);
   ttc(i) = (BF_ini(i)/UR(i));
   ttr(i) = (BF_max(i)*(1-record(step,i+1)))/(PR(i)-UR(i));
   chg_over(i) =  max(cost(i,:));
   tiq(i) = (BF_threshold(i)-record(step,i+1))*BF_max(i)/UR(i);%
end
alphap = 1/mean(ttc);
betap = 1/max(ttr); %
if max(tiq) > 0
   gamma_p = 1/max(tiq);
else
   gamma_p = 999999;
end
epsilon_p=1/max(chg_over);
```

## cur_state_fnx Function:

```
function [prod_selection,ans2_size,IX,cur_state,alpha_record,beta_record,gamma_record,epsilon_record,num_of_prod_sequenced,
      eta_record]=cur_state_selection_fnx(num_of_prod,step,record,time_stamp,BF_max,BF_ini,UR,PR,ans1_size,prod_thrs,selectio
      n_num,previous_prod,final_sequence)
global prod_selection,global ans2_size,global alpha,global beta,global gamma_,global epsilon, global eta,global cost;
global IX,global cur_state,global goodness_flag,global alpha_record,global beta_record,global gamma_record, global epsilon_record
global eta_record,global num_of_prod_sequenced, global milt_UR, global milt_UR_temp,global etap, global eta_val
seq_usage_rate(final_sequence,num_of_prod,UR);%calculate mlitenburg's U
prod_selection(1,1:num_of_prod)=0;
cur_state(step,1:num_of_prod)=0;
goodness_flag=0;
if ans1_size(1,2) == 1
   ans2_size(1,2)=1;
   for i = 1:num_of_prod
      if prod_thrs(i) > 0
```

```
            prod_selection(i)=1;
        end
    end
    return
else
    for i = 1:num_of_prod
        if prod_thrs(i) > 0
            if time_stamp(i) == 0
                prod_i=i;
                calc_milt_u_temp(final_sequence,prod_i,UR,step,num_of_prod,num_of_prod_sequenced,
                  milt_UR,etap,eta_val);
                cur_state(step,i) = -alpha*(BF_ini(i)/UR(i)) + beta*(BF_max(i)-BF_ini(i))/(PR(i)-UR(i))
                  - epsilon*cost(previous_prod,i) - eta * milt_UR_temp;
                %these next line are only for debugging, these values are not used for anything else
                alpha_record(step,i)=-alpha*(BF_ini(i)/UR(i));
                beta_record(step,i)=beta*(BF_max(i)-BF_ini(i))/(PR(i)-UR(i));
                gamma_record(step,i)=0;
                epsilon_record(step,i)=- epsilon*cost(previous_prod,i);
                eta_record(step,i)=-eta * milt_UR_temp;
            else
                prod_i=i;
                calc_milt_u_temp(final_sequence,prod_i,UR,step,num_of_prod,num_of_prod_sequenced,milt_UR,etap,eta_val);
                cur_state(step,i) = -alpha*(BF_ini(i)/UR(i)) + beta*((BF_max(i)-BF_ini(i))/(PR(i)-UR(i))) + gamma_*(record(step,1)
                  -time_stamp(i)) - epsilon*cost(previous_prod,i) - eta * milt_UR_temp;
                alpha_record(step,i)=-alpha*(BF_ini(i)/UR(i));
                beta_record(step,i)=beta*(BF_max(i)-BF_ini(i))/(PR(i)-UR(i));
                gamma_record(step,i)=gamma_*(record(step,1)-time_stamp(i));
                epsilon_record(step,i)=- epsilon*cost(previous_prod,i);
                eta_record(step,i)=-eta * milt_UR_temp;
            end
        else
            cur_state(step,i) = -inf;
            alpha_record(step,i)=0;
            beta_record(step,i)=0;
            gamma_record(step,i)=0;
            epsilon_record(step,i)=0;
        end
    end
end
[ranking,IX] = sort ((cur_state(step,:)),'descend');
goodness_flag=1;
index=0;
for i=1:num_of_prod
    if ranking(i) == 0
        index=1+index;
    else
        break
    end
end
prod_selection=IX;
ans2=find(prod_thrs == 1);
ans2_size=size(ans2);
```

## test_combined_sequence Function:

```
function [test_combined_sequence,seq_goodness]=test_combined_sequence(previous_prod,num_of_prod,lookahead_time,
            UR,PR,cost,cost_threshold,BF_threshold,BF_ini,BF_max,BF_tcross,prod_thrs,prod_selection,ans2_size,chi,psi);
global seq_goodness;global goodness_tr;global goodness_tr_time;global pg_num;global alpha, global beta,global gamma_, global
epsilon,global milt_UR,global eta_val, global num_of_prod_sequenced,global milt_UR_temp,global zeta_,global omega, global phi
goodness_tr=[];
goodness_tr_time=[];
chi=alpha;%lookahead time to crash weighting factor
psi=beta;%lookahead time to refill weighting factor
phi=gamma_;%lookahead time in queue weighting factor
omega=epsilon;%lookahead changeover cost weighting factor
la_step=size(num_of_prod_sequenced,1);
if la_step > 1
```

```
   num_of_prod_sequenced_LA=num_of_prod_sequenced(la_step-1,:);
else
   num_of_prod_sequenced_LA=num_of_prod_sequenced;
end
seq_goodness=[];
seq_goodness_stored=[];
goodness_BF_level=[];
seq_page_prev=1;
milt_UR_LA_max=1;
seq_flag=1;
goodness_percentage=2;%factor that the goodness can be below the previous goodness
decrs_gdns_flag=-1.1;
delta_t=1;
while seq_flag ~= 0
   goodness_tr_time(1,num_of_prod,2)=0;
   goodness_tr(1,num_of_prod,2)=0;
   seq_goodness(1,4+num_of_prod,2)=0;
   row =1;
   for a=1:num_of_prod;%this will list all the possible products that are to be considered for the lookahead function
      if (prod_thrs(a) == 1);
         seq_goodness(row,1)=0;%previous goodness value
         seq_goodness(row,2)=0;%current goodness calculated value
         seq_goodness(row,4)=a;%the 1st product of  the sequence
         goodness_tr(row,1)=seq_goodness(row,3)+(BF_max(a)-BF_ini(a)+(UR(a)*cost(previous_prod,a)))/(PR(a)-
              UR(a))+cost(previous_prod,a);%calc the refill time for prod a
         goodness_tr_time(row,a)=goodness_tr(row,1);%record time of product refill
         seq_goodness(row,3)=goodness_tr(row,1);%the time of  the sequence
         row=row+1;
      end;
   end;
   increment=row;
   for i=1:(increment - 1)%this is to calc the goodness for the sequence
      seq_loc=4;%the column number of the last product in the sequence
      last_prod=seq_goodness(i,seq_loc);
      seq_good=0;
      for j=1:num_of_prod
         if j==last_prod
            BF=BF_ini(j);
            seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
            BF = BF_max(j); %calculate buffer fullness for the last product that was refilled
         end
         if goodness_tr_time(i,j)==0
            BF = BF_ini(j)- UR(j) * goodness_tr_time(i,last_prod); %calculate buffer fullness for products that haven't been refilled
         else
            BF = BF_max(j) - UR(j) * (goodness_tr_time(i,last_prod) - goodness_tr_time(i,j)); %calculate buffer fullness for products
                                                                                                that have been refilled
         end
         goodness_BF_level(i,j)=BF;
         if BF <= 0; %
            seq_goodness(i,1)= -1; %label as bad b/c refill inventory is depleted
            j=num_of_prod;
         end
      end
      x=last_prod;
      calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
      seq_goodness(i,2)= seq_good - omega*cost(previous_prod,last_prod) -  zeta_ * milt_UR_temp;%this is the first goodness calc
      seq_goodness_stored(i,1)= seq_goodness(i,2);
      milt_UR_LA(i,1)=milt_UR_temp;%stores milt ur to be used for future LA steps
      num_of_prod_seqd_LA(i,:)=num_of_prod_sequenced_LA(1,:);%copies the initial numer of seqd products
      num_of_prod_seqd_LA(i,x)=num_of_prod_seqd_LA(i,x)+1;%adds the LA seqd product to the number of seqd products
   end
   seq_page=1;
   %
   %Step #2
   %
   seq_flag=0;
   row=1;i=1;
   seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
   seq_page=1+seq_page;%the sheet number of the seq_good variable
   la_step=la_step+1;%record step number for calculating milt_UR
```

```
while i<=(increment-1)
   seq_loc=5;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   if a > 0 && seq_goodness(i,3)>=0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(a,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,5,seq_page)=y;%the 2nd product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial num of seqd prd
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,
                     y,seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
         end
         i=i-1;
      end
      i=i+1;
   else
      if seq_goodness(i,1,seq_page_prev)== -2
         seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
         goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
         goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
         goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
         seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
         seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
         row = row+1;
      end
      i=i+1;
   end
end
if seq_flag == 0
   pg_num=seq_page-1;
   if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
      seq_page=pg_num;
   end
   if size(seq_goodness_stored,2)>1
      for i=1:(size(seq_goodness_stored,1))
         seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness
         seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness
         seq_goodness_stored(i,seq_page+3,seq_page)=i;
      end
   else
      seq_goodness_stored(num_of_prod,num_of_prod)=zeros;
   end
   break
end
```

```
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
   last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
   seq_good=0;
   x=seq_goodness(i,seq_loc,seq_page);%most recent product
   a=seq_goodness(i,4,seq_page_prev);%previously refilled product
   if x > 0 && seq_goodness(i,1,seq_page) >= 0
      for j=1:num_of_prod
         if j==x
            BF=goodness_BF_level(i,j,seq_page);
            seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
            BF = BF_max(j);
         end
         if goodness_tr_time(i,j,seq_page)==0
            BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
         else
            if goodness_tr_time(i,j,seq_page)>=lookahead_time
               BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                                                              that has been refilled, but time is greater than the lookahead time,
            else
               BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                                                                        for products that have been refilled
            end
         end
         goodness_BF_level(i,j,seq_page)=BF;
         if BF <= 0
            seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
            j=num_of_prod;
         end
      end
      milt_UR=milt_UR_LA(i,1,seq_page);
      num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                                 num_of_prod_sequenced_LA to be used by function
      calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
      seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
      milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
      num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                                 seqd products
      if seq_goodness(i,3,seq_page) >= lookahead_time
         seq_goodness(i,1,seq_page)=-2;
      end
      if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)%if the goodness for the last prod is not
                                                                 better or equal to the previous goodness, the seq is flagged as bad
         seq_goodness(i,1,seq_page)=decrs_gdns_flag;
         seq_goodness(i,2,seq_page)=seq_good;
      else
         seq_goodness_stored(i,seq_page,seq_page) = seq_good;
         seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
         seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
      end
   end
end
%
% Step #3
%
seq_flag=0;
seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
seq_page=1+seq_page;%the sheet number of the seq_good variable
la_step=la_step+1;%record step number for calculating milt_UR
row=1;i=1;
for i=1:(increment-1)
   seq_loc=6;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   x=seq_goodness(i,seq_loc-1,seq_page_prev);
   if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
```

```
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 2nd product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value being placed on current
                                                                          sheet in column 1
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial number of seqd
                                                                          products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                     seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
         end
         i=i-1;
      end
   else
      if seq_goodness(i,1,seq_page_prev)== -2
         seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
         goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
         goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
         goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
         goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
         seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
         seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
         row = row+1;
      end
   end
   i=i+1;
end
if seq_flag == 0
   pg_num=seq_page-1;
   if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
      seq_page=pg_num;
   end
   for i=1:size(seq_goodness_stored,1)
      seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
      seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
      seq_goodness_stored(i,seq_page+3,seq_page)=i;
   end
   break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
   last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
   seq_good=0;
   x=seq_goodness(i,seq_loc,seq_page);%most recent product
   if x > 0 && seq_goodness(i,1,seq_page)>= 0
      for j=1:num_of_prod
         if j==x
            BF=goodness_BF_level(i,j,seq_page);
```

```
            seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
            BF = BF_max(j);
         end
         if goodness_tr_time(i,j,seq_page)==0
            BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
         else
            if goodness_tr_time(i,j,seq_page)>=lookahead_time
               BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                                  that has been refilled, but time is greater than the lookahead time
            else
               BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                                  for products that have been refilled
            end
         end
         goodness_BF_level(i,j,seq_page)=BF;
         if BF <= 0
            seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
            j=num_of_prod;
         end
      end
      milt_UR=milt_UR_LA(i,1,seq_page);
      num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                                  num_of_prod_sequenced_LA to be used by function
      calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
      seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
      milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
      num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                                  seqd products
      if seq_goodness(i,3,seq_page) >= lookahead_time
         seq_goodness(i,1,seq_page)=-2;
      end
      if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                                  better or equl to the previous goodness, the seq is flagged as bad
         seq_goodness(i,1,seq_page)=decrs_gdns_flag;
         seq_goodness(i,2,seq_page)=seq_good;
      else
         seq_goodness_stored(i,seq_page,seq_page) = seq_good;
         seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
         seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
      end
   end
end
%
% Step #4
%
seq_flag=0;
seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
seq_page=1+seq_page;  %the sheet number of the seq_good variable
la_step=la_step+1; %record step number for calculating milt_UR
row=1;i=1;
while i<=(increment-1)
   seq_loc=7;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   b=seq_goodness(i,5,seq_page_prev);
   x=seq_goodness(i,seq_loc-1,seq_page_prev);
   if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-2,seq_page)=b;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 3rd product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 4th product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
```

```
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial numer of seqd
                                                products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                    seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
         end
         i=i-1;
      end
   else
      if seq_goodness(i,1,seq_page_prev)== -2
         seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
         goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
         goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
         goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
         seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
         seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
         row = row+1;
      end
   end
   i=i+1;
end
if seq_flag == 0
   pg_num=seq_page-1;
   if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
      seq_page=pg_num;
   end
   for i=1:size(seq_goodness_stored,1)
      seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
      seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
      seq_goodness_stored(i,seq_page+3,seq_page)=i;
   end
   break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
   last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
   seq_good=0;
   x=seq_goodness(i,seq_loc,seq_page);%most recent product
   if x > 0 && seq_goodness(i,1,seq_page)>= 0
      for j=1:num_of_prod
         if j==x
            BF=goodness_BF_level(i,j,seq_page);
            seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
            BF = BF_max(j);
         end
         if goodness_tr_time(i,j,seq_page)==0
            BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
         else
            if goodness_tr_time(i,j,seq_page)>=lookahead_time
               BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                   that has been refilled, but time is greater than the lookahead time
            else
               BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
```

```
                                                    for products that have been refilled
            end
          end
          goodness_BF_level(i,j,seq_page)=BF;
          if BF <= 0
            seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
            j=num_of_prod;
          end
        end
        milt_UR=milt_UR_LA(i,1,seq_page);
        num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                    num_of_prod_sequenced_LA to be used by function
        calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
        seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
        milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
        num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                    seqd products
        if seq_goodness(i,3,seq_page) >= lookahead_time
          seq_goodness(i,1,seq_page)=-2;
        end
        if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                    better or equl to the previous goodness, the seq is flagged as bad
          seq_goodness(i,1,seq_page)=decrs_gdns_flag;
          seq_goodness(i,2,seq_page)=seq_good;
        else
          seq_goodness_stored(i,seq_page,seq_page) = seq_good;
          seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
          seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
        end
      end
    end
    %
    % Step #5
    %
    seq_flag=0;
    seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
    seq_page=1+seq_page;  %the sheet number of the seq_good variable
    la_step=la_step+1; %record step number for calculating milt_UR
    row=1;i=1;
    while i<=(increment-1)
      seq_loc=8;%the column number of the last product in the sequence
      a=seq_goodness(i,4,seq_page_prev);
      b=seq_goodness(i,5,seq_page_prev);
      c=seq_goodness(i,6,seq_page_prev);
      x=seq_goodness(i,seq_loc-1,seq_page_prev);
      if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
        seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
        seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
        goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
        goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
        thres_flag=0;
        for y=1:num_of_prod;
          threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
          if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-3,seq_page)=b;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-2,seq_page)=c;%the 3rd product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 4th product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 5th product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial numer of seqd
                                                    products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                    seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
```

```
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
         end
         i=i-1;
      end
   else
      if seq_goodness(i,1,seq_page_prev)== -2
         seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
         goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
         goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
         goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
         seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
         seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
         row = row+1;
      end
   end
   i=i+1;
end

if seq_flag == 0
   pg_num=seq_page-1;
   if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
      seq_page=pg_num;
   end
   for i=1:size(seq_goodness_stored,1)
      seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
      seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
      seq_goodness_stored(i,seq_page+3,seq_page)=i;
   end
   break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
   last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
   seq_good=0;
   x=seq_goodness(i,seq_loc,seq_page);%most recent product
   if x > 0 && seq_goodness(i,1,seq_page)>= 0
      for j=1:num_of_prod
         if j==x
            BF=goodness_BF_level(i,j,seq_page);
            seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
            BF = BF_max(j);
         end
         if goodness_tr_time(i,j,seq_page)==0
            BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
         else
            if goodness_tr_time(i,j,seq_page)>=lookahead_time
               BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                                  that has been refilled, but time is greater than the lookahead time
            else
               BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                                  for products that have been refilled
            end
         end
         goodness_BF_level(i,j,seq_page)=BF;
         if BF <= 0
            seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
            j=num_of_prod;
         end
```

```
      end
      milt_UR=milt_UR_LA(i,1,seq_page);
      num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                    num_of_prod_sequenced_LA to be used by function
      calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
      seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
      milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
      num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                    seqd products
      if seq_goodness(i,3,seq_page) >= lookahead_time
         seq_goodness(i,1,seq_page)=-2;
      end
      if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                    better or equl to the previous goodness, the seq is flagged as bad
         seq_goodness(i,1,seq_page)=decrs_gdns_flag;
         seq_goodness(i,2,seq_page)=seq_good;
      else
         seq_goodness_stored(i,seq_page,seq_page) = seq_good;
         seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
         seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
      end
   end
end
%
% Step #6
%
seq_flag=0;
seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
seq_page=1+seq_page;  %the sheet number of the seq_good variable
la_step=la_step+1; %record step number for calculating milt_UR
row=1;i=1;
while i<=(increment-1)
   seq_loc=9;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   b=seq_goodness(i,5,seq_page_prev);
   c=seq_goodness(i,6,seq_page_prev);
   d=seq_goodness(i,7,seq_page_prev);
   x=seq_goodness(i,seq_loc-1,seq_page_prev);
   if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-4,seq_page)=b;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-3,seq_page)=c;%the 3rd product of  the sequence
            seq_goodness(row,seq_loc-2,seq_page)=d;%the 4th product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 5th product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 6th product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial numer of seqd
                                                    products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                     seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
```

```
            end
        end
        if thres_flag==0
            seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
            for j=1:num_of_prod
                goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
            end
            i=i-1;
        end
    else
        if seq_goodness(i,1,seq_page_prev)== -2
            seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
            row = row+1;
        end
    end
    i=i+1;
end
if seq_flag == 0
    pg_num=seq_page-1;
    if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
        seq_page=pg_num;
    end
    for i=1:size(seq_goodness_stored,1)
        seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
        seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
        seq_goodness_stored(i,seq_page+3,seq_page)=i;
    end
    break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
    last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
    seq_good=0;
    x=seq_goodness(i,seq_loc,seq_page);%most recent product
    if x > 0 && seq_goodness(i,1,seq_page)>= 0
        for j=1:num_of_prod
            if j==x
                BF=goodness_BF_level(i,j,seq_page);
                seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
                BF = BF_max(j);
            end
            if goodness_tr_time(i,j,seq_page)==0
                BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
            else
                if goodness_tr_time(i,j,seq_page)>=lookahead_time
                    BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                                    that has been refilled, but time is greater than the lookahead time
                else
                    BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                                    for products that have been refilled
                end
            end
            goodness_BF_level(i,j,seq_page)=BF;
            if BF <= 0
                seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
                j=num_of_prod;
            end
        end
        milt_UR=milt_UR_LA(i,1,seq_page);
        num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                                    num_of_prod_sequenced_LA to be used by function
        calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
        seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
        milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
```

```
      num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                    seqd products
      if seq_goodness(i,3,seq_page) >= lookahead_time
         seq_goodness(i,1,seq_page)=-2;
      end
      if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                    better or equl to the previous goodness, the seq is flagged as bad
         seq_goodness(i,1,seq_page)=decrs_gdns_flag;
         seq_goodness(i,2,seq_page)=seq_good;
      else
         seq_goodness_stored(i,seq_page,seq_page) = seq_good;
         seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
         seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
      end
   end
end
%
% Step #7
%
seq_flag=0;
seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
seq_page=1+seq_page;  %the sheet number of the seq_good variable
la_step=la_step+1; %record step number for calculating milt_UR
row=1;i=1;
while i<=(increment-1)
   seq_loc=10;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   b=seq_goodness(i,5,seq_page_prev);
   c=seq_goodness(i,6,seq_page_prev);
   d=seq_goodness(i,7,seq_page_prev);
   e=seq_goodness(i,8,seq_page_prev);
   x=seq_goodness(i,seq_loc-1,seq_page_prev);
   if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-5,seq_page)=b;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-4,seq_page)=c;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-3,seq_page)=d;%the 3rd product of  the sequence
            seq_goodness(row,seq_loc-2,seq_page)=e;%the 4th product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 5th product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 6th product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial numer of seqd
                                                    products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                     seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
```

```
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
        end
        i=i-1;
    end
  else
    if seq_goodness(i,1,seq_page_prev)== -2
        seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
        goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
        goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
        goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
        seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
        seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
        row = row+1;
    end
  end
  i=i+1;
end
if seq_flag == 0
  pg_num=seq_page-1;
  if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
    seq_page=pg_num;
  end
  for i=1:size(seq_goodness_stored,1)
    seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
    seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
    seq_goodness_stored(i,seq_page+3,seq_page)=i;
  end
  break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
  last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
  seq_good=0;
  x=seq_goodness(i,seq_loc,seq_page);%most recent product
  if x > 0 && seq_goodness(i,1,seq_page)>= 0
    for j=1:num_of_prod
      if j==x
        BF=goodness_BF_level(i,j,seq_page);
        seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
        BF = BF_max(j);
      end
      if goodness_tr_time(i,j,seq_page)==0
        BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
      else
        if goodness_tr_time(i,j,seq_page)>=lookahead_time
          BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                    that has been refilled, but time is greater than the lookahead time
        else
          BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                     for products that have been refilled
        end
      end
      goodness_BF_level(i,j,seq_page)=BF;
      if BF <= 0
        seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
        j=num_of_prod;
      end
    end
    milt_UR=milt_UR_LA(i,1,seq_page);
    num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                    num_of_prod_sequenced_LA to be used by function
    calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
    seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
    milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
    num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                    seqd products
    if seq_goodness(i,3,seq_page) >= lookahead_time
      seq_goodness(i,1,seq_page)=-2;
    end
```

```
      if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                          better or equl to the previous goodness, the seq is flagged as bad
         seq_goodness(i,1,seq_page)=decrs_gdns_flag;
         seq_goodness(i,2,seq_page)=seq_good;
      else
         seq_goodness_stored(i,seq_page,seq_page) = seq_good;
         seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
         seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
      end
   end
end
%
% Step #8
%
seq_flag=0;
seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
seq_page=1+seq_page;  %the sheet number of the seq_good variable
la_step=la_step+1; %record step number for calculating milt_UR
row=1;i=1;
while i<=(increment-1)
   seq_loc=11;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   b=seq_goodness(i,5,seq_page_prev);
   c=seq_goodness(i,6,seq_page_prev);
   d=seq_goodness(i,7,seq_page_prev);
   e=seq_goodness(i,8,seq_page_prev);
   f=seq_goodness(i,9,seq_page_prev);
   x=seq_goodness(i,seq_loc-1,seq_page_prev);
   if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-6,seq_page)=b;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-5,seq_page)=c;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-4,seq_page)=d;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-3,seq_page)=e;%the 3rd product of  the sequence
            seq_goodness(row,seq_loc-2,seq_page)=f;%the 4th product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 5th product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 6th product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial numer of seqd
                                                          products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                     seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
         end
         i=i-1;
```

```
        end
    else
        if seq_goodness(i,1,seq_page_prev)== -2
            seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
            row = row+1;
        end
    end
    i=i+1;
end
if seq_flag == 0
    pg_num=seq_page-1;
    if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
        seq_page=pg_num;
    end
    for i=1:size(seq_goodness_stored,1)
        seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
        seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
        seq_goodness_stored(i,seq_page+3,seq_page)=i;
    end
    break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
    last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
    seq_good=0;
    x=seq_goodness(i,seq_loc,seq_page);%most recent product
    if x > 0 && seq_goodness(i,1,seq_page)>= 0
        for j=1:num_of_prod
            if j==x
                BF=goodness_BF_level(i,j,seq_page);
                seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
                BF = BF_max(j);
            end
            if goodness_tr_time(i,j,seq_page)==0
                BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
            else
                if goodness_tr_time(i,j,seq_page)>=lookahead_time
                    BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                                that has been refilled, but time is greater than the lookahead time
                else
                    BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                                for products that have been refilled
                end
            end
            goodness_BF_level(i,j,seq_page)=BF;
            if BF <= 0
                seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
                j=num_of_prod;
            end
        end
        milt_UR=milt_UR_LA(i,1,seq_page);
        num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                                num_of_prod_sequenced_LA to be used by function
        calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
        seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
        milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
        num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                                seqd products
        if seq_goodness(i,3,seq_page) >= lookahead_time
            seq_goodness(i,1,seq_page)=-2;
        end
        if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                                better or equl to the previous goodness, the seq is flagged as bad
            seq_goodness(i,1,seq_page)=decrs_gdns_flag;
```

```
        seq_goodness(i,2,seq_page)=seq_good;
      else
        seq_goodness_stored(i,seq_page,seq_page) = seq_good;
        seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
        seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
      end
    end
end
%
% Step #9
%
seq_flag=0;
seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
seq_page=1+seq_page  %the sheet number of the seq_good variable
la_step=la_step+1; %record step number for calculating milt_UR
row=1;i=1;
while i<=(increment-1)
   seq_loc=12;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   b=seq_goodness(i,5,seq_page_prev);
   c=seq_goodness(i,6,seq_page_prev);
   d=seq_goodness(i,7,seq_page_prev);
   e=seq_goodness(i,8,seq_page_prev);
   f=seq_goodness(i,9,seq_page_prev);
   g=seq_goodness(i,10,seq_page_prev);
   x=seq_goodness(i,seq_loc-1,seq_page_prev);
   if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-7,seq_page)=b;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-6,seq_page)=c;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-5,seq_page)=d;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-4,seq_page)=e;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-3,seq_page)=f;%the 3rd product of  the sequence
            seq_goodness(row,seq_loc-2,seq_page)=g;%the 4th product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 5th product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 6th product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial numer of seqd
                                                products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                  seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
         end
         i=i-1;
      end
```

```
    else
        if seq_goodness(i,1,seq_page_prev)== -2
            seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
            row = row+1;
        end
    end
    i=i+1;
end
if seq_flag == 0
    pg_num=seq_page-1;
    if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
        seq_page=pg_num;
    end
    for i=1:size(seq_goodness_stored,1)
        seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
        seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
        seq_goodness_stored(i,seq_page+3,seq_page)=i;
    end
    break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
    last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
    seq_good=0;
    x=seq_goodness(i,seq_loc,seq_page);%most recent product
    if x > 0 && seq_goodness(i,1,seq_page)>= 0
        for j=1:num_of_prod
            if j==x
                BF=goodness_BF_level(i,j,seq_page);
                seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
                BF = BF_max(j);
            end
            if goodness_tr_time(i,j,seq_page)==0
                BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
            else
                if goodness_tr_time(i,j,seq_page)>=lookahead_time
                    BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                                that has been refilled, but time is greater than the lookahead time
                else
                    BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                                for products that have been refilled
                end
            end
            goodness_BF_level(i,j,seq_page)=BF;
            if BF <= 0
                seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
                j=num_of_prod;
            end
        end
        milt_UR=milt_UR_LA(i,1,seq_page);
        num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                                num_of_prod_sequenced_LA to be used by function
        calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
        seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
        milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
        num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                                seqd products
        if seq_goodness(i,3,seq_page) >= lookahead_time
            seq_goodness(i,1,seq_page)=-2;
        end
        if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                                better or equl to the previous goodness, the seq is flagged as bad
            seq_goodness(i,1,seq_page)=decrs_gdns_flag;
            seq_goodness(i,2,seq_page)=seq_good;
```

```
      else
         seq_goodness_stored(i,seq_page,seq_page) = seq_good;
         seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
         seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
      end
   end
end
%
% Step #10
%
seq_flag=0;
seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
seq_page=1+seq_page;  %the sheet number of the seq_good variable
la_step=la_step+1; %record step number for calculating milt_UR
row=1;i=1;
while i<=(increment-1)
   seq_loc=13;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   b=seq_goodness(i,5,seq_page_prev);
   c=seq_goodness(i,6,seq_page_prev);
   d=seq_goodness(i,7,seq_page_prev);
   e=seq_goodness(i,8,seq_page_prev);
   f=seq_goodness(i,9,seq_page_prev);
   g=seq_goodness(i,10,seq_page_prev);
   h=seq_goodness(i,11,seq_page_prev);
   x=seq_goodness(i,seq_loc-1,seq_page_prev);
   if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-8,seq_page)=b;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-7,seq_page)=c;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-6,seq_page)=d;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-5,seq_page)=e;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-4,seq_page)=f;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-3,seq_page)=g;%the 3rd product of  the sequence
            seq_goodness(row,seq_loc-2,seq_page)=h;%the 4th product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 5th product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 6th product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial numer of seqd
                                                products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                  seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
         end
         i=i-1;
```

```
      end
   else
      if seq_goodness(i,1,seq_page_prev)== -2
         seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
         goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
         goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
         goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
         seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
         seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
         row = row+1;
      end
   end
   i=i+1;
end
if seq_flag == 0
   pg_num=seq_page-1;
   if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
      seq_page=pg_num;
   end
   for i=1:size(seq_goodness_stored,1)
      seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
      seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
      seq_goodness_stored(i,seq_page+3,seq_page)=i;
   end
   break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
   last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
   seq_good=0;
   x=seq_goodness(i,seq_loc,seq_page);%most recent product
   if x > 0 && seq_goodness(i,1,seq_page)>= 0
      for j=1:num_of_prod
         if j==x
            BF=goodness_BF_level(i,j,seq_page);
            seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
            BF = BF_max(j);
         end
         if goodness_tr_time(i,j,seq_page)==0
            BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
         else
            if goodness_tr_time(i,j,seq_page)>=lookahead_time
               BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                                  that has been refilled, but time is greater than the lookahead time
            else
               BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                                  for products that have been refilled
            end
         end
         goodness_BF_level(i,j,seq_page)=BF;
         if BF <= 0
            seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
            j=num_of_prod;
         end
      end
      milt_UR=milt_UR_LA(i,1,seq_page);
      num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                                  num_of_prod_sequenced_LA to be used by function
      calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
      seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
      milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
      num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                                  seqd products
      if seq_goodness(i,3,seq_page) >= lookahead_time
         seq_goodness(i,1,seq_page)=-2;
      end
      if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                                  better or equl to the previous goodness, the seq is flagged as bad
         seq_goodness(i,1,seq_page)=decrs_gdns_flag;
```

```
        seq_goodness(i,2,seq_page)=seq_good;
      else
        seq_goodness_stored(i,seq_page,seq_page) = seq_good;
        seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
        seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
      end
    end
end
%
% Step #11
%
seq_flag=0;
seq_page_prev=seq_page;%the previous sheet number of the seq_good variable
seq_page=1+seq_page  %the sheet number of the seq_good variable
la_step=la_step+1; %record step number for calculating milt_UR
row=1;i=1;
while i<=(increment-1)
   seq_loc=14;%the column number of the last product in the sequence
   a=seq_goodness(i,4,seq_page_prev);
   b=seq_goodness(i,5,seq_page_prev);
   c=seq_goodness(i,6,seq_page_prev);
   d=seq_goodness(i,7,seq_page_prev);
   e=seq_goodness(i,8,seq_page_prev);
   f=seq_goodness(i,9,seq_page_prev);
   g=seq_goodness(i,10,seq_page_prev);
   h=seq_goodness(i,11,seq_page_prev);
   m=seq_goodness(i,12,seq_page_prev);
   x=seq_goodness(i,seq_loc-1,seq_page_prev);
   if x > 0 && seq_goodness(i,1,seq_page_prev) >= 0
      seq_goodness(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      seq_goodness_stored(row+num_of_prod*1,seq_page+1,seq_page)=0;
      goodness_tr(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      goodness_tr_time(row+num_of_prod*1,num_of_prod*1,seq_page)=0;
      thres_flag=0;
      for y=1:num_of_prod;
         threshold_percentage=goodness_BF_level(i,y,seq_page_prev)/BF_max(y);
         if (cost(x,y) <= cost_threshold && BF_threshold(y) >= threshold_percentage );
            seq_goodness(row,4,seq_page)=a;%the 1st product of  the sequence
            seq_goodness(row,seq_loc-9,seq_page)=b;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-8,seq_page)=c;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-7,seq_page)=d;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-6,seq_page)=e;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-5,seq_page)=f;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-4,seq_page)=g;%the 2nd product of  the sequence
            seq_goodness(row,seq_loc-3,seq_page)=h;%the 3rd product of  the sequence
            seq_goodness(row,seq_loc-2,seq_page)=m;%the 4th product of  the sequence
            seq_goodness(row,seq_loc-1,seq_page)=x;%the 5th product of  the sequence
            seq_goodness(row,seq_loc,seq_page)=y;%the 6th product of  the sequence
            seq_goodness(row,2,seq_page)=seq_goodness(i,2,seq_page_prev);%previous goodness value
            seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
            goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
            goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
            goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
            milt_UR_LA(row,1,seq_page)=milt_UR_LA(i,1,seq_page_prev);
            num_of_prod_seqd_LA(row,:,seq_page)=num_of_prod_seqd_LA(i,:,seq_page_prev);%copies the initial numer of seqd
                                                products
            goodness_tr(row,seq_loc-3,seq_page)=seq_goodness(i,3,seq_page_prev)+(BF_max(y)-goodness_BF_level(row,y,
                  seq_page)+UR(y)*cost(x,y))/(PR(y)-UR(y))+cost(x,y);
            goodness_tr_time(row,y,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%flag product as being refilled previously
            seq_goodness(row,3,seq_page)=goodness_tr(row,seq_loc-3,seq_page);%the total time of  the sequence
            row = row+1;
            if goodness_tr(row,seq_loc-3,seq_page) < lookahead_time
               seq_flag=1;%used to flag system to know that at least one sequence hasn't reach lookahead time yet
            end
            thres_flag=1;
         end
      end
      if thres_flag==0
         seq_goodness(i,3,seq_page_prev)=seq_goodness(i,3,seq_page_prev)+delta_t;
         for j=1:num_of_prod
```

```
            goodness_BF_level(i,j,seq_page_prev)=goodness_BF_level(i,j,seq_page_prev)-UR(j)*delta_t;
        end
        i=i-1;
    end
  else
    if seq_goodness(i,1,seq_page_prev)== -2
        seq_goodness(row,:,seq_page)=seq_goodness(i,:,seq_page_prev);%
        goodness_BF_level(row,:,seq_page)=goodness_BF_level(i,:,seq_page_prev);
        goodness_tr(row,:,seq_page)=goodness_tr(i,:,seq_page_prev);
        goodness_tr_time(row,:,seq_page)=goodness_tr_time(i,:,seq_page_prev);%flag product as being refilled previously
        seq_goodness_stored(row,:,seq_page) = seq_goodness_stored(i,:,seq_page_prev);
        seq_goodness_stored(row,seq_page+1,seq_page) = seq_goodness_stored(row,seq_page,seq_page);
        row = row+1;
    end
  end
  i=i+1;
end
if seq_flag == 0
  pg_num=seq_page-1;
  if min(seq_goodness(:,1,pg_num))>-2 && max(seq_goodness(:,1,pg_num))<=0
    seq_page=pg_num;
  end
  for i=1:size(seq_goodness_stored,1)
    seq_goodness_stored(i,seq_page+1,seq_page)=min(seq_goodness_stored(i,1:seq_page,seq_page));%store min goodness val
    seq_goodness_stored(i,seq_page+2,seq_page)=max(seq_goodness_stored(i,1:seq_page,seq_page));%store max goodness val
    seq_goodness_stored(i,seq_page+3,seq_page)=i;
  end
  break
end
milt_UR_LA_max=max(1,max(milt_UR_LA(seq_page_prev)));
increment=row;
for i=1:(increment - 1)%this is to calc the goodness for the sequence
  last_prod=seq_goodness(i,seq_loc-1,seq_page);%previous product
  seq_good=0;
  x=seq_goodness(i,seq_loc,seq_page);%most recent product
  if x > 0 && seq_goodness(i,1,seq_page)>= 0
    for j=1:num_of_prod
      if j==x
        BF=goodness_BF_level(i,j,seq_page);
        seq_good= - chi*BF/UR(j) + psi*(BF_max(j)-BF)/(PR(j)-UR(j)) + phi*max(0,(BF_max(j)*BF_threshold(j) - BF)/UR(j));
        BF = BF_max(j);
      end
      if goodness_tr_time(i,j,seq_page)==0
        BF = BF_ini(j)- UR(j) * seq_goodness(i,3,seq_page); %calculate buffer fullness for products that haven't been refilled
      else
        if goodness_tr_time(i,j,seq_page)>=lookahead_time
          BF = BF_max(j) - PR(j)*[lookahead_time - goodness_tr(i,seq_loc-4,seq_page)]; %calculate buffer fullness for product
                                                    that has been refilled, but time is greater than the lookahead time
        else
          BF = BF_max(j) - UR(j) * (seq_goodness(i,3,seq_page) - goodness_tr_time(i,j,seq_page)); %calculate buffer fullness
                                                    for products that have been refilled
        end
      end
      goodness_BF_level(i,j,seq_page)=BF;
      if BF <= 0
        seq_goodness(i,1,seq_page)= -1; %label as bad b/c refill inventory is depleted
        j=num_of_prod;
      end
    end
    milt_UR=milt_UR_LA(i,1,seq_page);
    num_of_prod_sequenced_LA=num_of_prod_seqd_LA(i,:,seq_page);%copies number of seqd products into
                                                    num_of_prod_sequenced_LA to be used by function
    calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,eta_val,milt_UR_LA_max);
    seq_good = seq_good - omega*cost(last_prod,x) -  zeta_ * milt_UR_temp;%this is the first goodness calc
    milt_UR_LA(i,1,seq_page)=milt_UR_temp;%stores milt_ur of current seqn to be used for future LA steps of the seqn
    num_of_prod_seqd_LA(i,x,seq_page)=num_of_prod_seqd_LA(i,x,seq_page)+1;%adds the LA seqd product to the number of
                                                    seqd products
    if seq_goodness(i,3,seq_page) >= lookahead_time
      seq_goodness(i,1,seq_page)=-2;
    end
```

```
        if (seq_goodness(i,2,seq_page)-seq_good)/abs(seq_good) >= (goodness_percentage)  %if the goodness for the last prod is not
                                                                    better or equl to the previous goodness, the seq is flagged as bad
           seq_goodness(i,1,seq_page)=decrs_gdns_flag;
           seq_goodness(i,2,seq_page)=seq_good;
        else
           seq_goodness_stored(i,seq_page,seq_page) = seq_good;
           seq_goodness_stored(i,seq_page+1,seq_page) = mean(seq_goodness_stored(i,1:seq_page,seq_page));
           seq_goodness(i,2,seq_page)=seq_goodness_stored(i,seq_page+1,seq_page);
        end
     end
  end
  seq_flag=0;
end
```

## seq_usage_rate Function:

```
function [milt_UR,num_of_prod_sequenced]=seq_usage_rate(final_sequence,num_of_prod,UR)
global milt_UR, global total_demand,global record, global num_of_prod_sequenced
num_of_prod_sequenced=[];
num_of_prod_sequenced(length(final_sequence)+1,num_of_prod)=0;
%calc of the total number of products produced for each stage of the
%sequence, used for Miltenberg's euation
if length(final_sequence) >= 1
  for k = 1:length(final_sequence)
     for i = 1:num_of_prod
        if final_sequence(k,1) == i
           num_of_prod_sequenced(k+1,i)=num_of_prod_sequenced(k,i)+1;
        else
           num_of_prod_sequenced(k+1,i)=num_of_prod_sequenced(k,i);
        end
     end
  end
else
  final_sequence=0;
end
if size(num_of_prod_sequenced,1) > 2
  num_of_prod_sequenced(1:2,:)=[];
end
milt_UR = 0;
for k = 1:length(final_sequence)-1
  for i = 1:num_of_prod
     milt_UR=(num_of_prod_sequenced(k,i) - k * (UR(i)/sum(UR(1:num_of_prod))))^2+milt_UR;
  end
end
```

## calc_milt_u_temp Function:

```
function [milt_UR_temp,
eta]=calc_milt_u_temp(final_sequence,prod_i,UR,step,num_of_prod,num_of_prod_sequenced,milt_UR,etap,eta_val)
global milt_UR_temp, global eta%, global eta_val
if step > 1
  stage=step-1;
else
  stage=1;
end
milt_UR_temp=0;
for i = 1:num_of_prod
  if i == prod_i
     milt_UR_temp=(num_of_prod_sequenced(stage,i)+1 - stage * (UR(i)/sum(UR(1:num_of_prod))))^2+milt_UR_temp;
  else
     milt_UR_temp=(num_of_prod_sequenced(stage,i) - stage * (UR(i)/sum(UR(1:num_of_prod))))^2+milt_UR_temp;
  end
end
if stage > 1
  milt_UR_temp=milt_UR_temp+milt_UR;
end
```

```
if milt_UR ~= 0
   etap=1/milt_UR;
else
   etap=0;
end
eta = etap*eta_val;
```

## calc_milt_u_temp_LA Function:

```
function [milt_UR_temp,zeta_]=calc_milt_u_temp_LA(x,UR,la_step,num_of_prod,num_of_prod_sequenced_LA,milt_UR,
   eta_val,milt_UR_LA_max)
global milt_UR_temp, global zeta_%, global eta_val
prod_i=x;
stage=la_step;
milt_UR_temp=0;
for z = 1:num_of_prod
   if z == prod_i %if current product is being refilled, then number of prods is increased by 1
      milt_UR_temp=(num_of_prod_sequenced_LA(z)+1 - stage * (UR(z)/sum(UR(1:num_of_prod))))^2+milt_UR_temp;
   else
      milt_UR_temp=(num_of_prod_sequenced_LA(z) - stage * (UR(z)/sum(UR(1:num_of_prod))))^2+milt_UR_temp;
   end
end
milt_UR_temp=milt_UR_temp+milt_UR;
zetap=1/milt_UR_LA_max;
zeta_ = zetap*eta_val;
```

## Appendix III: A History of Production Control Systems

**A History of Production Control Systems**

This appendix will provide general information and the history of the many different methods of scheduling production, such as Classical Scheduling, Bottleneck Scheduling, MRP, Pull Production Control, and Diagnostic Scheduling.

**Classical Scheduling**

Scheduling of jobs for manufacturing is as old as manufacturing itself but research in the area dates back to the early 1900s with the emergence of the scientific management movement. The use of computers in the 1950s and 1960s allowed researchers to begin to perform serious analysis of scheduling problems and led to the eventual development of MRP. Classical scheduling problems were examined before computers or MRP and many assumptions were required in order to be able to solve these problems by hand. Some of the more common assumptions include: all jobs are available at the start of the problem, process times are deterministic, process times are independent from the schedule (no setup times), machines do not break down, and there is no preemption or cancellation of jobs. These assumptions allow many of the scheduling problems to be reduced down to a manageable task. In some cases these assumptions allow a sequence to be found instead of an actual schedule, but in other cases a full-blown schedule is needed which is much more difficult to determine.

Classical scheduling problems are often highly simplified and generic but they can offer useful insights into scheduling issues even though most problems only address one, two, or possibly three machines. An example of insight gained from classical scheduling is when a single machine is examined using the previously mentioned assumptions and the average cycle time is minimized; where cycle time is defined as the average time from the release of a job until it reaches an inventory point and the end of the routing. It is interesting to note that although the total time to complete multiple jobs is independent of the sequence, the average cycle time is minimized by processing jobs in order of their processing times, starting with the shortest one first. This is called the shortest process time (SPT) sequencing rule. This problem shows that congestion in a factory can be reduced by processing shorter jobs first because they will move through the factory quicker and not block the longer processing time jobs. Askin [87] provided a visual representation of this in the following chart, where the horizontal axis is the part number and the vertical axis is the manufacturing time required to process the job. This shows how each subsequent job is delayed by the preceding job and the height of the column is the total time required to process job *n*.

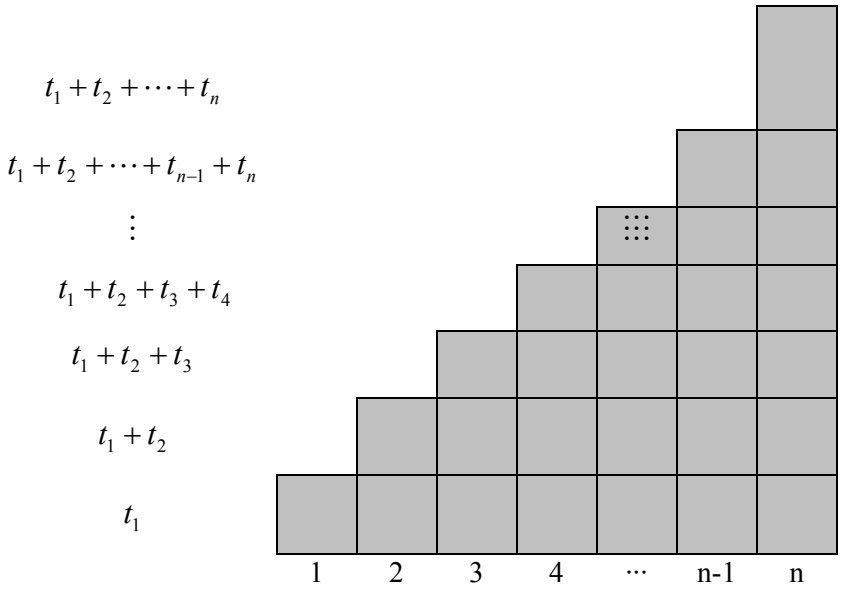

Figure 1: Flow Time Delays

The earliest due date (EDD) sequencing is another insight that was gained from classical scheduling. The maximum lateness of any job was shown to be minimized by ordering the jobs according to their due dates, with the earliest due date first and the latest due date last. The idea behind this sequencing technique is that all the jobs will be completed on time if it is possible.

The minimization of the makespan, which is the total time to finish all jobs on two machines, is another area studied by classical scheduling. The makespan time is dependent upon the order of the jobs because certain sequences could cause idle time at the second machine while it waits for the job on the first machine to be completed, while other sequences could cause the first machine to wait while the second machine completes a job. Johnson [9] developed an algorithm to find a sequence that minimizes makespan for this problem. The algorithm is as follows:

> Separate the job into two sets, A and B. Jobs in set A are those whose process time on the first machine is less than or equal to the process time on the second machine. Set B contains the remaining jobs. The processing sequence begins with set A in the order of shortest processing time first. Set B is processed after the completion of set A and the order of the jobs in set B begin with the longest processing time first.

The insight from this algorithm is that by processing the shortest processing times in the beginning of the sequence allows the wait time for the second machine to be minimized because the second machine is idle until the first job is completed on the first machine. The ending

sequence also helps to minimize the total processing time because the first machine will be idle as the second machine finishes the last job.

Many of the assumptions used in classical scheduling are not true for most real-world manufacturing systems. There are almost always more than three machines to schedule parts on in the real-world which makes classical scheduling infeasible. Also, real-world processing times may appear to be deterministic but there will always be some randomness and variability which can cause a great deal of congestion in manufacturing systems. The assumption that all jobs are ready at the beginning is far from accurate because jobs continue to arrive during the life of a manufacturing system and the system is not completely emptied prior to starting a new job. Another incorrect assumption is that process time is independent of the schedule because the sequence in which parts are scheduled can have a significant impact on the time to get through the plant because of setup times. Time can often be saved if similar parts that share the same setup or a similar setup are scheduled together.

Another problem with classical scheduling is the time required to determine an optimal schedule as the number of jobs is in the hundreds and the number of machines is in the tens. The classical algorithms can be divided into two categories: (1) Class P problems that can be solved by algorithms whose computational time grows as a polynomial function of the size of the problem. (2) NP-hard problems are those for which no polynomial algorithm exists and computational time increases exponentially with the size of the problem. Hopp and Spearman [88] provided the following comparison between the two types of problems. Suppose we want to determine the optimum sequence for twenty jobs on a single machine (NP-hard problem, 20! possibilities) and we use a computer that can examine 1,000,000 sequences per second. This problem would take 77,147 years to complete the sequencing problem, while ten jobs would take only 3.63 seconds. An example of a Class P problem would be to sort jobs for a processing center according to processing time. Consider sorting twenty jobs and using a computer that can sort ten jobs in the same amount of time as sequencing ten jobs. The sorting of twenty jobs would take only 9.4 seconds. The sorting of jobs can be accomplished by many well known algorithms for sorting a list of elements whose computation time is proportional to $n$ log $n$, where $n$ is the number of elements sorted. This algorithm is bounded by $n^2$, a polynomial.

Most real world scheduling problems fall into the NP-hard category of problems and tend to be large and difficult to solve. Often it is impossible to solve actual problems due to the required

computational time. Fortunately a good solution can be found even though it is perhaps not the optimal solution. The nonpolynomial nature of the problems implies that there are many solutions which can actually help to find a good solution. Consider that if only one in a trillion of the possible solutions is good in the previous twenty job scheduling problem, there would still be almost 2.5 million good solutions.

Problem size and lack of an efficient solution scheme often lead to the use of a heuristic to find a good, nearly optimal, solution to the problem. The solution that is found may or may not be the optimum and even if it is optimal, there will be no way to confirm the solution. Bartholdi and Platzman [89] gave the following heuristic summary:

> A heuristic may be viewed as an information processor that deliberately but judiciously ignores certain information. By ignoring information, a heuristic is freed from whatever effort might have been required to read the data and compute with it. Moreover, the solution produced by such a heuristic is independent of the ignored information, and thus unaffected by changes in that information. Of course the art of heuristic design lies in knowing exactly what information to ignore. Ideally, one seeks to ignore information that is expensive to gather and maintain, that is computationally expensive to exploit, and that contributes little additional accuracy to the solution.

Heuristics must be used to find a solution to the very difficult real world problems because of the limited time in a real manufacturing setting and the need for a good solution that does not necessarily have to be the optimum. A good heuristic will accurately represent the manufacturing system and omit superfluous information that would only slow down the solution.

**Bottleneck Scheduling**

The scheduling of multiple machines and multiple products can be a very difficult if not impossible task but it is often simplified by focusing on the bottleneck and using insight gained from classical scheduling techniques for a single process. A bottleneck is defined as the process that takes the largest amount of time when compared to all the other processes required to make a product. The bottleneck is the critical process that will determine the rate at which material will flow through all the other processes. Once the bottleneck is sequenced, then the remaining machines are scheduled to prevent starvation by preceding processes and to prevent blocking of the bottleneck by subsequent processes.

The Theory of Constraints (TOC) is a management philosophy that was developed by Eliyahu Goldratt and it focuses on both financial and operational measures to determine the performance

of a manufacturing operation. TOC helps the company achieve its goal of making money by identifying the limitations due to the constraints in three areas: internal resources, market demand, and the company's policies. There are nine rules that are used to direct the management of the factory floor (the internal resources):

1. Balance flow in the factory, not capacity. -- The flow of the product is more important than maintaining a high utilization of the capacity at each process, as long as there is sufficient capacity.
2. Constraints determine the utilization of the non-bottleneck processes. -- Maximizing utilization of non-bottleneck processes only increases costs, idle time, and resources.
3. Activity is not equivalent to utilization. -- Producing output that cannot fit into the schedule of the bottleneck only increases the inventory. A busy machine is not equivalent to a productive machine that is meeting the required workload.
4. An hour lost at the bottleneck is an hour lost for the entire system. -- This causes the bottleneck to be scheduled at or near full utilization in order to maximize the total production of the system. This rule also encourages reduction in setups, setup times, and the use of large lots at the bottleneck.
5. An hour saved at a non-bottleneck is a mirage. -- If the time is saved prior to the bottleneck the product will still be required to wait the hour that was saved in order to fit into the original schedule of the bottleneck. However it is possible that if time can be saved at subsequent processes, the throughput time will be decreased because the product will leave the system quicker.
6. The bottleneck governs both throughput and inventory of the system. -- The production rate of the bottleneck determines the throughput time and inventory (Little's Law).
7. The transfer batch to the next process should not always be equal to the process batch. -- It may be advantageous to transfer smaller batches between non-bottleneck processes in order to decrease idle time.
8. The process batch size should be variable and not fixed. -- The process batch size depends on the current state of the system and most often should be large at the bottleneck and smaller elsewhere.
9. The schedule should be determined by examining all the constraints simultaneously. -- Lead times are a function of the schedule and cannot be predetermined and will often be variable dependent upon the state of the system.

The TOC is somewhat intuitive but the implementation on the shop floor is perhaps slightly more difficult to grasp. The Drum-Buffer-Rope (DBR) is a production control technique that is used for non-bottleneck processes to produce only enough to keep the bottleneck at the capacity required to meet demand. The DBR technique can be thought of as a group of people traveling, each at different rates. To keep everyone together a rope is connected between each person which will cause everyone to travel at the rate of the slowest person.

The drum represents two possibilities, the bottleneck or the market demand, and will be responsible for "striking the beat" to set the pace for the entire system similar to the cadence set by the percussion section for a marching band. The buffers are used to ensure that the cadence is maintained at critical points in the manufacturing system. The buffers should be placed before and after the bottleneck and at shipping to ensure that due dates are met. The real world demand and the bottleneck are tied together with the first rope and the bottleneck and the raw material release point are tied together with the second rope. The rope also serves to constrain the WIP levels such as if the bottleneck machine breaks down, the upstream processes will not keep sending products downstream. The DBR configuration will regulate the overall production rate so that the customer demand will be met and that the bottleneck will not be starved or overloaded with products from upstream processes.

**MRP**

Material Requirement Planning (MRP) has become widely used in industry to determine schedules for manufacturing systems and it was one of the first attempts to assist production planning with the use of computers. MRP coordinates the release of orders to push the items through the production processes in an attempt to minimize any unnecessary inventory. The decision making process for MRP relies on the use of inventory records, bill of material structures, the master production schedule, and lead time estimates.

MRP operates on a continuous horizon of time periods, or time buckets, that most often represent up to one week of production. For each period inventory status is updated and the bill of material (BOM) for each finished product is exploded down the product hierarchy one level at a time. Current inventory and open orders are matched against gross requirements at each level over the planning horizon and then order releases are planned to meet the remaining net requirements. The planned order releases for parent items become the gross requirements at the next lower level of the hierarchy. This is accomplished by using the following steps:

1. Netting: The net requirements are determined by subtracting on-hand inventory and scheduled receipts from the gross requirements.
2. Lot sizing: The netted demand is divided into appropriate lot sizes.
3. Time phasing: The due date of each job is offset by the lead time to determine the start time.
4. BOM explosion: The BOM is used to determine the gross requirements for required components at the next level.
5. Iterate: Repeat steps until all levels have been scheduled.

To illustrate these steps, suppose a bicycle was being scheduled for production and it had the following BOM and 50 finished bikes are required for week number four and week number six. Currently 25 bikes are on-hand and the lead time for all components is one week.

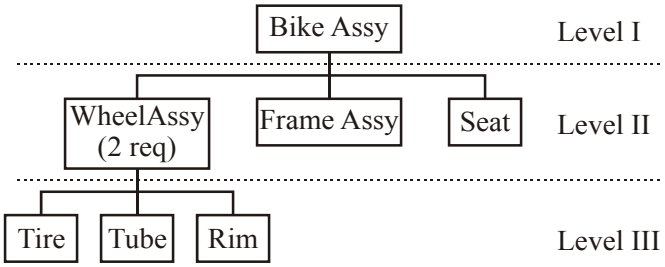


Figure 2: Bike BOM

Table 1.1: MRP Input Data

| **Bike Assembly** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | | 50 | | 50 |
| Scheduled Receipts | | | | | | | |
| Projected On-hand | 25 | | | | | | |
| Net Requirements | | | | | | | |
| Planned Order Releases | | | | | | | |

**Netting:**

The first step is to begin the netting process which computes the net demand. Most MRP systems assume that the coverage of demand will come first from on-hand inventory, second from scheduled receipts, and finally from planned order releases. The amount of coverage of the demand, $D_t$, by the on-hand inventory, $I_t$, and the scheduled receipt, $S_t$, is calculated using the following equation where $I_0$ is equal to the initial on-hand inventory.

$$I_t = I_{t-1} + S_t - D_t$$

This equation is used for every time period until the demand becomes greater than the on-hand inventory and scheduled receipts are able to cover.

The net requirements for period $t$, $N_t$, will be zero for an period in which the on-hand inventory is greater than zero ($N_t = 0$ if $I_t \geq 0$). The net requirements for a period in which the on-hand inventory is less than zero is equal to the absolute value of the on-hand inventory ($N_t = |I_t|$ if $I_t < 0$). This can be seen in the following table.

Table 1.2: Updated Net Requirements

| **Bike Assembly** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | | 50 | | 50 |
| Scheduled Receipts | | | | | | | |
| Projected On-hand | 25 | 25 | 25 | 25 | -25 | 0 | -50 |
| Net Requirements | | | | | 25 | | 50 |
| Planned Order Releases | | | | | | | |

**Lot Sizing:**

This step involves determining how to dispatch jobs to meet the net requirements of the production schedule. The simplest lot-sizing rule is known as lot for lot, and it states that the amount to be produced in a period is equal to the net requirements for the period. This technique requires no thought or calculations and is similar to the just-in-time philosophy of making only what is needed. For this example, 50 products would be released to meet the Week 6 demand and 25 products to meet the Week 4 demand.

Another simple lot-sizing rule is known as fixed order period (FOP) or period order quantity. This rule is an attempt to reduce the number of setups by combining the requirements of P periods and releasing the multiple requirements at the same time. FOP becomes lot for lot when P is equal to one. In this example, suppose P = 3, the net requirements for weeks 4, 5, and 6 would all be released together as an order for 75 products.

Fixed order quantity and economic order quantity (EOQ) are along the same line of thinking as FOP except that a predetermined quantity is released when an order is required. This is practical because often fixtures, carts, and other similar devices in a factory can hold only a set quantity of the product. For instance, many more seats can be transported than bike frames. EOQ is an attempt to find the optimum order quantity when minimizing setup costs (A) and inventory holding costs. EOQ can provide an order quantity although it is based on flawed assumptions such as constant demand (D) and no relationship between inventory cost (h) and batch size. EOQ is given by the following equation:

$$Q = \sqrt{\frac{2AD}{h}}$$

**Time Phasing:**

This step is used to offset the release date of jobs by the lead time from when the net requirements are actually required. The standard assumption used by MRP is that the lead time of a job is a constant value that is independent of the size of the job or the status of the shop floor. These assumptions potentially lead to problems that will be discussed later. The equation used to offset the planned order releases for time $t, POR_t$, is shown below, where the lead time is represented by $\tau$.

$$POR_t = N_{t+\tau}$$

The lead times for this example are assumed to be one week for the bike assembly and assuming the use of lot for lot replenishment rule, the net requirements for Week 4 would be offset by one week to Week 3. The same would be done for the net requirements in Week 6, which would be offset to Week 5; the updated results are shown below. Notice that the lot size for Week 5 is double that of Week 3, but both lots are assumed to be completed in one time period.

Table 1.3: Updated Lot Sizing and Time Phasing

| **Bike Assembly** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | | 50 | | 50 |
| Scheduled Receipts | | | | | | | |
| Projected On-hand | 25 | 25 | 25 | 25 | -25 | 0 | -50 |
| Net Requirements | | | | | 25 | | 50 |
| Planned Order Releases | | | | 25 | | 50 | |

**BOM Explosion:**

The purpose of this step is to transfer the release dates and lot sizes from the current level to the next lower level. In this example the release dates and lot sizes for the top level (Level I) assembly will be transferred to the Level II components that are required to make the finished assembly. From the BOM, two wheel assemblies, one frame assembly, and one seat are required for every bike. The updated BOM explosion is shown in the tables below for each of the three components. These gross requirements would be added to any other accumulated demand for the components, such as if the wheel assembly in this example was required for a tandem bike assembly or some other finished assembly.

Table 1.4: Wheel Assembly BOM Explosion

| **Wheel Assembly** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | 50 | | 100 | |
| Scheduled Receipts | | | 25 | | | | |
| Projected On-hand | 15 | | | | | | |
| Net Requirements | | | | | | | |
| Planned Order Releases | | | | | | | |

Table 1.5: Frame Assembly BOM Explosion

| **Frame Assembly** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | 25 | | 50 | |
| Scheduled Receipts | | | | | | | |
| Projected On-hand | 5 | | | | | | |
| Net Requirements | | | | | | | |
| Planned Order Releases | | | | | | | |

Table 1.6: Seat BOM Explosion

| **Seat** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | 25 | | 50 | |
| Scheduled Receipts | | | | | | | |
| Projected On-hand | 35 | | | | | | |
| Net Requirements | | | | | | | |
| Planned Order Releases | | | | | | | |

**Iterate:**

The iteration step is the final step in the MRP process and is used to repeat the previous steps for all the subsequent levels of required components of the final assembly. For this example the wheel assembly will be iterated to the lowest level (Level III), at which point all the components required will be scheduled. The updated MRP table for the wheel assembly is shown below after the netting, lot sizing, and time phasing steps have been completed.

Table 1.7: Updated Wheel Assembly

| **Wheel Assembly** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | 50 | | 100 | |
| Scheduled Receipts | | | 25 | | | | |
| Projected On-hand | 15 | 15 | 40 | -10 | 0 | -100 | |
| Net Requirements | | | | 10 | | 100 | |
| Planned Order Releases | | | 10 | | 100 | | |

Table 1.8: Updated Frame Assembly

| **Frame Assembly** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | 25 | | 50 | |
| Scheduled Receipts | | | | | | | |
| Projected On-hand | 5 | 5 | 5 | -20 | 0 | -50 | |
| Net Requirements | | | | 20 | | 50 | |
| Planned Order Releases | | | 20 | | 50 | | |

Table 1.9: Updated Seat

| **Seat** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | | 25 | | 50 | |
| Scheduled Receipts | | | | | | | |
| Projected On-hand | 35 | 35 | 35 | 10 | 10 | -40 | |
| Net Requirements | | | | | | 40 | |
| Planned Order Releases | | | | | 40 | | |

At this point the BOM explosion step is required for each of the Level II components. The gross requirements for the components of the wheel assembly will have the following MRP tables after the BOM explosion step.

Table 1.10: Input Tire Data

| **Tire** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | 10 | | 100 | | |
| Scheduled Receipts | | 15 | | | | | |
| Projected On-hand | 40 | | | | | | |
| Net Requirements | | | | | | | |
| Planned Order Releases | | | | | | | |

Table 1.11: Input Tube Data

| **Tube** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | 10 | | 100 | | |
| Scheduled Receipts | | | | 30 | | | |
| Projected On-hand | 30 | | | | | | |
| Net Requirements | | | | | | | |
| Planned Order Releases | | | | | | | |

Table 1.12: Input Rim Data

| **Rim** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | 10 | | 100 | | |
| Scheduled Receipts | | | 20 | | | | |
| Projected On-hand | 20 | | | | | | |
| Net Requirements | | | | | | | |
| Planned Order Releases | | | | | | | |

Once again the iteration of the previous steps is required to complete the production schedule for all the components required by the wheel assembly. The final updated tables for each of the components are shown below. At this point the MRP process is complete, but the process will need to be repeated for the current and subsequent levels if any of the values change for the gross requirements or scheduled receipts at any level in the BOM.

Table 1.13: Input Tire Data

| **Tire** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | 10 | | 100 | | |
| Scheduled Receipts | | 15 | | | | | |
| Projected On-hand | 40 | 55 | 45 | 45 | -55 | - | - |
| Net Requirements | | | | | 55 | | |
| Planned Order Releases | | | | 55 | | | |

Table 1.14: Input Tube Data

| **Tube** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | 10 | | 100 | | |
| Scheduled Receipts | | | | 30 | | | |
| Projected On-hand | 30 | 30 | 20 | 50 | -50 | | |
| Net Requirements | | | | | 50 | | |
| Planned Order Releases | | | | 50 | | | |

Table 1.15: Input Rim Data

| **Rim** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 |
|---|---|---|---|---|---|---|---|
| Gross Requirements | | | 10 | | 100 | | |
| Scheduled Receipts | | | 20 | | | | |
| Projected On-hand | 20 | 20 | 30 | 30 | -70 | | |
| Net Requirements | | | | | 70 | | |
| Planned Order Releases | | | | 70 | | | |

**Lead times:**

A problem with MRP is the use of the assumption that constant lead times accurately represent manufacturing lead times which actually vary continually due to randomness. Lead time for a product is composed of the actual processing time on the machine, queuing delays, waiting time during machine breakdowns, etc. To compensate for the variation in lead times, the user will often choose long pessimistic estimates for lead times because excess inventory is viewed as less critical than having a late job and missing the customer's due date. For example, suppose the average manufacturing lead time is three weeks with a standard deviation of one week. The customer due date will be met if the planned lead time is chosen to be five weeks, but the majority of the jobs will be held in inventory for two weeks. As planned lead times are increased, the waiting time for the next operation also increases which in turn causes the system to have more inventory. In the previous example, the majority of the jobs will be held in inventory for three weeks on average. Lead times must be increased for busy or near capacity production in order to compensate for the increased queuing delays in order to ensure that the product will still meet the due date of the customer. The increase in lead times causes the inventory levels to increase which then causes the cycle times to further increase.

The use of the assumption of constant lead times no matter the size of the job implies the assumption that the processing center has infinite capacity. To illustrate how these two assumptions are intertwined with one another, imagine that the process is for a person to cross from one side of a highway, without any traffic, to the other. If one person or fifty people try to cross the road, there will be enough room (infinite capacity) for everyone to cross at the same time so that the lead time is truly constant no matter the number of people which represent the batch or lot size. However, a queue would be formed if everyone had to cross the road at the crosswalk and there would be some delay as people would wait for their turn to cross the road. The crosswalk represents a finite capacity process, where perhaps only four people can cross at

the same time, which is what occurs in real world manufacturing systems.

**Capacity infeasibility:**

The basic methodology and the assumption of MRP is that of a production line with a fixed lead time that is independent of the workload in the plant. This means that MRP assumes that there will always be sufficient capacity on the line no matter how great the demand. This assumption is much too simplistic to be effective and accurate at production scheduling. As the plant approaches operation at or near capacity this assumption becomes inaccurate and will cause customer due dates to be missed and inventory levels to increase.

Rough-cut capacity planning (RCCP) and capacity requirements planning (CRP) were two techniques developed to address capacity problems and to detect scheduling infeasibilities in MRP. RCCP is used prior to MRP to detect capacity violations resulting from the master production schedule (MPS). RCCP uses the bill of resources or bill of capacity, which gives the number of hours required at each critical resource to build a particular finished product as well as all exploded requirements, to check for infeasibilities of each finished product. RCCP ignores lead times, lot sizes, and inventory status and merely translates the demand over time into a profile of capacity requirements by multiplying the number of hours required by the product at a resource times the demand for the product. As an example to help understand RCCP, suppose part A and part B are made on the same machine and part A is composed of part $A_1$ and part $A_2$. Part A requires one hour of processing time on the machine, while part $A_1$ requires half an hour of processing time and part $A_2$ requires one hour and part B requires 2 hours of processing time. The bill of resources for these two parts is as follows:

Table 1.16: Bill of Resources

| Part A | Part B |
|---|---|
| 2.5 | 2 |

Table 1.17: Demand

| Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Part A | 10 | 10 | 10 | 20 | 20 | 20 | 20 | 10 |
| Part B | 5 | 25 | 5 | 15 | 10 | 25 | 15 | 5 |

Based on the demand above for part A and part B the RCCP calculations will be as follows:

Table 1.18: RCCP Calculations

| Week | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Total (hours) |
|---|---|---|---|---|---|---|---|---|---|
| Part A (hours) | 25 | 25 | 25 | 50 | 50 | 50 | 50 | 25 | 300 |
| Part B (hours) | 10 | 50 | 10 | 30 | 20 | 50 | 30 | 10 | 210 |
| Total | 35 | 75 | 35 | 80 | 70 | 100 | 80 | 35 | 510 |
| Available | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 65 | 520 |
| Over(+)/Under(-) | 30 | -10 | 30 | -15 | -5 | -35 | -15 | 30 | - |

From these calculations, it is obvious that there are times in which there is insufficient capacity. Although if one only considers the eight time periods in aggregate there appears to be an excess of ten hours of production capacity. The planner must decide how to correct the time periods with capacity infeasibilities because RCCP only notifies the planner of a problem and does nothing to correct the situation. It is also important to note that RCCP assumes that the part and all subassemblies are processed on the same machine during the same time period and the processing time is independent of job sequence.

CRP provides a more detailed capacity check on an MRP produced schedule than RCCP. CRP requires the knowledge of planned order releases, existing WIP positions, routing data, capacity, and lead times for all resources in order to calculate the capacity required for the remaining orders (after WIP and inventories have been subtracted). CRP assumes that the time to go through a machine does not change even when the load exceeds capacity. CRP is only a good predictor of loading conditions in the very near term and it does not generate finite capacity analysis. CRP performs infinite forward loading, which means that it uses fixed lead times to predict job completion times for each process center and then computes a predicted loading over time. The predicted loadings are then compared against the available capacity, but no corrections are made for overload situations; therefore, all estimates of CRP beyond an overloaded condition are erroneous.

To better illustrate how CRP works, consider a processing center with a capacity of 400 parts per day, a three day lead time, and with no work in the processing center at the start. Also assume that the schedule uses a lot-sizing rule to reduce setups and the planned order releases are for 1,200 units on Day 1 and Day 4. CRP would produce a load profile for the processing center that has an overload condition on Day 3 and Day 6 and no production for the other days. A very

different result would be found if a finite capacity loading analysis were performed. The results would show that there is no output for the first two days as the product works through the processing center, but on Day 3 there would be an output of 400 units and this would continue for the next six days. The second release of 1,200 units would arrive at the processing center just as the last 400 units of the previous order are pulled into the processing center.

**Nervousness:**

Nervousness in an MRP system occurs when a small change in the master production schedule (MPS) causes a significant change in the planned order releases. Vollmann et al. [90] demonstrate an example in which a small decrease in demand in the MPS can cause a formerly feasible MRP plan to become infeasible. For this example, consider two parts. Part A has a two week lead time and uses FOP lot-sizing rule with an order period of five weeks and requires one unit of Part B. Part B has a lead time of four weeks and uses the FOP lot-sizing rule with a period of five weeks. The initial MRP calculations are shown before, prior to a change in demand.

Table 1.19: MRP Calculations for Part A

| **Part A** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 |
|---|---|---|---|---|---|---|---|---|---|
| Gross Requirements | | 2 | 24 | 3 | 5 | 1 | 3 | 4 | 50 |
| Scheduled Receipts | | | | | | | | | |
| Projected On-hand | 28 | 26 | 2 | -1 | -5 | -1 | -3 | -4 | -50 |
| Net Requirements | | | | 1 | 5 | 1 | 3 | 4 | 50 |
| Planned Order Releases | | 14 | | | | | 50 | | |

Table 1.20: MRP Calculations for Part B

| **Part B** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 |
|---|---|---|---|---|---|---|---|---|---|
| Gross Requirements | | 14 | | | | | 50 | | |
| Scheduled Receipts | | 14 | | | | | | | |
| Projected On-hand | 2 | 2 | 2 | 2 | 2 | 2 | -48 | | |
| Net Requirements | | | | | | | 48 | | |
| Planned Order Releases | | | 48 | | | | | | |

The nervousness of the system becomes apparent when the demand for Part A is decreased in

Week 2 from 24 units to 23 units. One would think that a schedule that is feasible for 24 units should also be feasible for a decreased demand of 23 units in the same period, but notice what happens in the MRP calculations shown below.

Table 1.21: Updated MRP Calculations for Part A

| **Part A** | | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 |
|---|---|---|---|---|---|---|---|---|---|
| Gross Requirements | | 2 | **23** | 3 | 5 | 1 | 3 | 4 | 50 |
| Scheduled Receipts | | | | | | | | | |
| Projected On-hand | 28 | 26 | 3 | 0 | -5 | -1 | -3 | -4 | -50 |
| Net Requirements | | | | | 5 | 1 | 3 | 4 | 50 |
| Planned Order Releases | | | **63** | | | | | | |

The original planned order releases were scheduled for 14 units during Week 1 and 50 units during Week 6, but because of the decrease in demand by one unit 63 units are scheduled to be released during Week 2. This would be a major change in the production schedule for a manufacturer who is attempting to meet the due dates of many customers at the same time in a dynamic environment where customer demand might fluctuate from week to week.

Vollmann suggests the use of different lot-sizing rules for each level in the BOM, with fixed order quantities for finished products, lot-for-lot or fixed order quantity for intermediate levels, and fixed order period for the lower levels. Another technique to decrease the nervousness of the system is to freeze the beginning of the MPS. This is very effective because changes in the near production order releases are the most disruptive and this method creates a frozen zone in which the first X time periods cannot be modified.

**MRP Production Control:**

MRP has been used by many companies over the past few decades as a production control system to schedule production and determine when to order materials to supply the production. MRP is useful to determine material requirements from outside the manufacturing system such as for subcomponents from vendors or orders for raw materials. MRP fails when it is used to schedule release periods for production on the shop floor because it operates independent of the status of jobs on the floor. MRP uses the push philosophy of moving products through the different stages of production and if there is any problem during production the system will fill up with WIP.

MRP is much like running water into a bathtub because there is no connection between the faucet and the drain. Once the faucet is turned on, water will continue to run whether the drain is open or if it is plugged; the user must be aware of the system and determine the amount of water flow to keep the tub from overflowing.

MRP provides no information to the scheduler regarding bottleneck processes, whether it is being starved, overloaded, or if it is dynamic and moves from one station to another. Classical scheduling techniques have been seen as providing valuable information for the scheduling of bottlenecks.

**Pull Production Systems:**

The just-in-time (JIT) or pull production method was very successfully implemented by the Japanese over the past several decades. This method is somewhat similar to the DBR in the sense that the customer demand and the rope is what control the production upstream from the finished products and the WIP is capped. JIT differs from other production methods in that it is a philosophy for the entire production system as much as it is a production scheduling technique. JIT is referred to as a pull production method because the material is pulled downstream as the customer removes a finished product from the company. Pull production systems will process a component to replace stock, meaning the component that was taken by the customer who could be the downstream process or an external customer removing finished goods. A push system processes a component to meet an order whether it is forecasted demand or a "hard" order from a customer.

The JIT or pull production method has been very successfully implemented by the Japanese over the past several decades. JIT differs from other production methods in that it is a philosophy for the entire production system as much as it is a production scheduling technique. JIT is referred to as a pull production method because information flows upstream as the material is pulled downstream when the customer removes a finished product from the company.

The pull production method used as a production scheduler has many advantages over conventional push methods. The first and largest advantage is that production under pull conditions actually is dependent upon the condition of the shopfloor. MRP uses estimated lead times and timed releases of jobs and it assumes that the product will flow through the factory smoothly and lead times must be inflated to account for disruptions. Conversely, with pull

production jobs are released only when there is customer demand and the downstream station authorizes the preceding station to begin work. If there is a machine that has a breakdown then the upstream machines that feed the broken machine will sit idle until the breakdown is fixed. This implies that the WIP in a pull system is capped and it will never exceed the initial line loading. The controlled status of WIP between adjacent processes also allows for strategic planning to help smooth product flow, such as ensuring that a static bottleneck is never starved or blocked. Often pull production will not be able to function well if the bottleneck is dynamic and moves around the factory floor. The controlled lower WIP status of a pull system allows for faster throughput times, lower holding costs, and improved quality controls. The high level of WIP in push production can cover many problems that become evident in a pull system, similar to exposing rocks hidden at the bottom of a stream by lowering the water level. The lower WIP levels of pull decrease the variability in lead times and lessen the need for inflated safety lead times that are often used with MRP. An MRP system will likely experience problems with starvation of processes when operated at low levels of WIP and will experience a WIP explosion when operating near full capacity; neither of these problems occur with pull production.

A pull system when implemented on the shopfloor can take on many different forms, including the following: a storefront system, kanban system, two-card kanban system, route-specific kanban system, CONWIP line, tandem CONWIP line, or a hybrid push/pull or pull/pull system. None of these pull systems can be deemed superior to the rest because each system will work well when matched correctly with the appropriate manufacturing process.

**Storefront or Supermarket Pull System:**

Pull systems have been functioning for decades in the form of grocery stores where the customer removes the goods from the shelves. This system can have many different looks, but typically in manufacturing environments a space is designated for one or more finished products on the outbound side of the process, production is signaled to begin when a space becomes open. This type of a pull system is referred to as a storefront or supermarket pull system and it works very well in manufacturing environments with low product variety because the storefront for a process will require one or more products in the storefront for every variant that is required for the downstream process. A storefront system will require very high levels of WIP in each storefront for a manufacturer that produces hundreds of different products.

**Kanban Pull System:**

The kanban pull system gets its name from the work authorization signal, which is issued by a kanban, which is a Japanese word and loosely translates as card and is used to govern the flow of materials through the plant. Production is triggered by the demand of a part being removed from the final inventory point. The kanban is removed from the finished product that the customer is taking and given to the preceding workstation at which point the work station is given authorization to replace the part that has been removed. This workstation then removes a part from the input buffer and sends the kanban upstream to the previous workstation to replace the part that was just used by the downstream station. This is propagated up the production line authorizing each workstation to replenish the void that was created by the demand. The authorizations flow in the opposite direction of the flow of material and pull the job through the stages of production.

**Signal Kanban Pull System:**

The signal kanban is a pull system that is often used when production involves significant time or costs associated with setups [8]. The signal kanban system uses a single signal or authorization for production, called a signal kanban. Each product has a specific level of inventory and the signal is released to replenish the inventory when the inventory falls below that level. The signal enters into the queue at the process, which creates a sequenced production at the process. Often this system is used when the process has significant setups or minimum batch sizes. This system was in use at the metal stamping line at the Toyota manufacturing plant in Georgetown, Kentucky. A more detail discussion of this system can be found in Monden [91].

**Two-Card Kanban Pull System:**

The two-card kanban pull system differs with the previously discussed kanban system in that there are two kanban cards used to authorize production and the workstations are not collocated. Examination of the following diagram will clarify how this system is used on the shopfloor. Workstation B has two containers of parts that have been removed by the customer, shown by the dashed outline of each container. The kanban card for each container is then placed in production kanban container, where one production card is still located (Steps 1 and 2). The operator of Workstation B will then remove the production kanban from the production kanban container and verify that the parts required to replenish the kanban are present at the station (Step 3). If the required parts are not present then the operator will move to the next kanban card in FIFO order until the production kanban and required parts are both available at the workstation. The operator

then removes the transport kanban and places it in the transportation kanban container (Step 4) where it will wait until it is picked up by the transporter who will carry the transportation kanban to Workstation A (Step 6). The operator of Workstation B moves the inbound container of parts needed to replenish the outbound production kanban, which is represented in the diagram by the dashed outline of the inbound container (Step 5). Upon arrival at Workstation A the transporter will remove the container of parts specified on the transportation kanban and place the attached production kanban in the production kanban container at Workstation A (Steps 7 and 8). The transporter will attach the transportation kanban on the parts container (Step 9) and transport the container back to Workstation B where it will be placed in the inbound stock buffer (Step 10). The production kanban at Workstation A will authorize the replenishment of the removed container and this will start the same process of production and transportation from the preceding workstation.

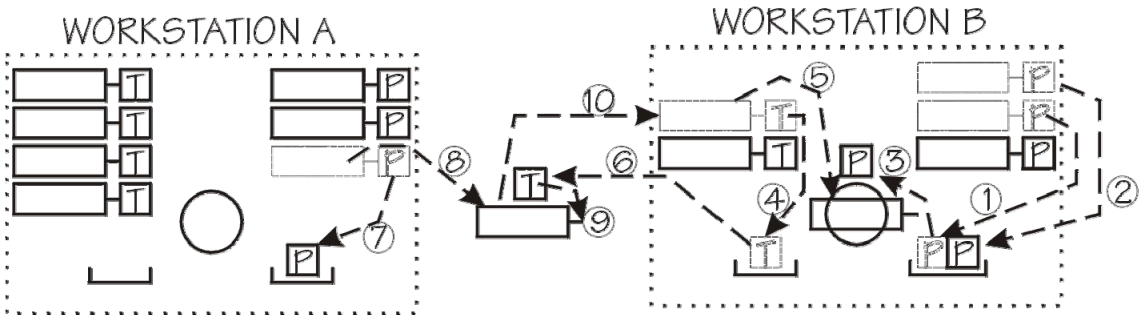


Figure 3: Two-Card Kanban Pull System

**Route Specific Kanbans:**

The issue of stock aggregation and the type of kanban that is used to manage the product variety is often solved by using route specific kanbans. A segregated inventory accommodates variation for each variant individually while an aggregated inventory will accommodate variation for all variants combined into the whole system. A kanban can be issued for a specific part or a specific route, as shown in the diagram below. The difference in the inventory levels at each workstation is very apparent. The part-specific kanban (top diagram) will have a much higher WIP level than the route-specific kanban system. A build list is used to control the variation in the route-specific kanban system and the type of part being built can be one of two parts for the top route and one of three parts for the bottom route. The route specific method can handle a large number of product variations, as well as more demand variability with a given WIP level. A supplemental source of information is required to define what product to build for a route-specific kanban system.

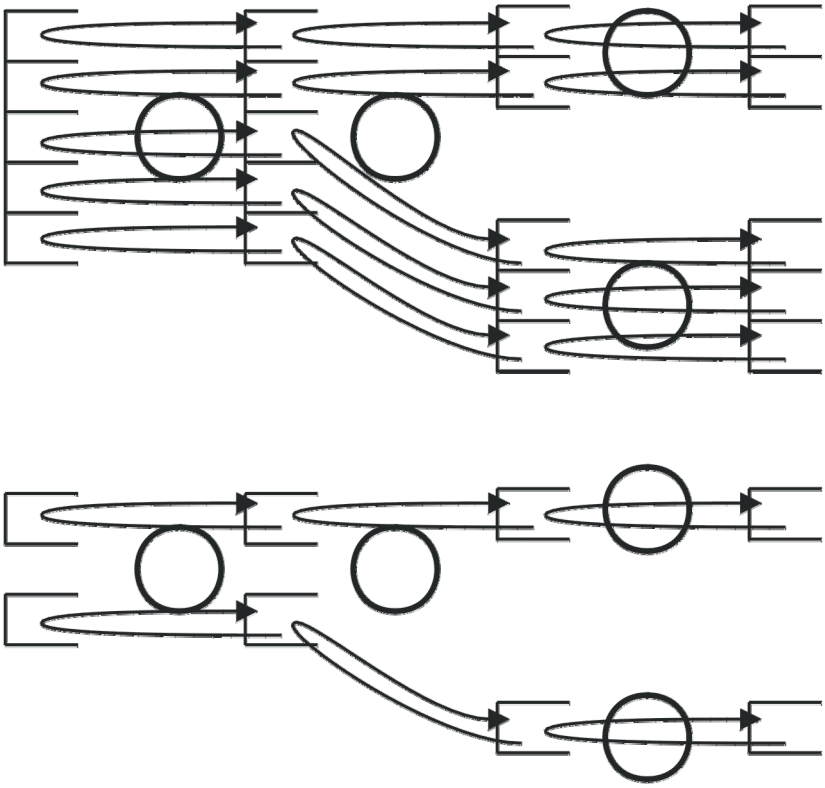

Figure 4: Part-Specific versus Route-Specific Kanban Systems

The route-specific kanban system works very similar to the previously discussed kanban systems in that a card authorizes work and the product is pulled through the various stages. There are different ways in which the product variation can be controlled, such as using a build list at each stage to control which variant to build. This option can become very difficult to manage due to the requirement of parts arriving and leaving each stage at a predicted time. Another option for controlling the product variation is to use a build list at an early stage to define the build order for all subsequent stages and kanban will pull the products through production in FIFO order. This control method is similar to the CONWIP pull system.

**CONWIP Pull System:**

CONWIP, constant work in progress, is a technique comparable to kanban systems but it uses loops on one or more machines to limit the amount of WIP. Each job will have a card associated with it and when a job exits the loop the card will be freed and sent to the beginning of the loop. The freed card at the beginning of the loop authorizes the release of another job into the loop.

The released job may be different than the job that just exited the system and depends upon the build list or sequence to control the variants. The released job is simply pushed through the CONWIP loop using the first-in-system, first-out (FISFO) processing rule. This causes some to not classify this production system as a pull production system, but for this research a pull system is defined as a system with capped WIP and information flow in the opposite direction of material flow.

A CONWIP system will perform very similar to a route-specific kanban system although a card could authorize work for several workstations instead of a single workstation. If a disruption occurs in a CONWIP system the products downstream will continue to flow out of the loop and new jobs will be released into the system and all the jobs will eventually accumulate at the disruption if it lasts long enough. All the jobs will also eventually accumulate behind the bottleneck if it is stationary and the system runs for an extended amount of time.

**Hybrid Pull Systems:**

There are differing opinions of how to design an appropriate hybrid pull system according to research that has been conducted over the past decade. Some researchers define a hybrid system as a mixture of CONWIP loops and kanban loops, referred to as a hybrid pull/pull system, which one could argue is actually a Tandem CONWIP system with some single machine loops included with larger CONWIP loops. Other researchers believe that an effective hybrid system, referred to as a hybrid push/pull system, is one in which the material is pushed into the system and the product is pulled through processing and assembly stages. Other researchers (and the final assembly line at the Toyota plant in Georgetown, Kentucky) believe that an effective hybrid system is one that pulls products through the initial stages and push products through the final stages. The point at which the push/pull interface occurs is often referred to as a junction point or a pacemaker and it is the point where product variety significantly increases.

**Diagnostic Scheduling**

Diagnostic scheduling is a technique that evolved from MRP over the past few decades as researchers began to realize that effective scheduling is far more advanced than merely finding a solution to mathematical problems. The key advantage of diagnostic scheduling is that it allows the user to determine the best solution when taking into account factors that cannot be evaluated by a computer such as: increasing capacity by adding overtime, the use of temporary workers, pushing back due dates for certain jobs or splitting jobs for customers that can tolerate missed due

dates. Ideally diagnostic scheduling will provide a feasible solution but if one is not available, possible changes to make the schedule feasible will be suggested to the user.

The two types of schedule infeasibility that diagnostic scheduling will predict are WIP infeasibility and capacity infeasibility. WIP infeasibility is caused by WIP that is inappropriately positioned, the only solution for finished products that are near the due date is to push or postpone the demand. Infinite capacity would be of no help to solve WIP infeasibility because the WIP is not where it needs to be in order to produce the finished product. Capacity infeasibility is caused by lacking sufficient capacity for a given job and it can be corrected by pushing out the demand or by increasing the capacity by means such as adding overtime.

To illustrate this, suppose for example, that Part A is built from two units of Part B at the rate of 150 parts per day. The demand for Part A is 100 units on day one, 100 units on day two, and 200 units on day three with 100 units of Part B available on day one, 500 on day two, and 200 units of Part B available on day three. One can quickly see that only 50 units of Part A can be produced on day one because of the lack of units of Part B. To fix this WIP infeasibility, the only solution is to postpone 50 units of Part A to day two where there is sufficient capacity to produce 150 units of Part A and a sufficient number of available units of Part B. Day two has sufficient capacity and 200 extra units of Part B that can be carried over to Day 3 in the form of available WIP in order to meet the requirement of 400 units of Part B. The demand for Day 3 is greater than the capacity, therefore it is capacity infeasible, but this can be corrected by postponing the production of 50 units of Part A until Day 4 or increase capacity for Day 3 by 50 units.

A further extension of diagnostic scheduling is capacitated material requirements planning (MRP-C), which was developed for a Doctoral dissertation by Tardif [92]. This is a procedure to determine a manufacturing schedule and during the scheduling process it will detect and remedy infeasibilities and allow user intervention. MRP-C begins by determining a low-level schedule that meets all due dates without exceeding capacity and builds the minimum possible amount of inventory before it is needed, i.e. build-ahead inventory. The objective of MRP-C is to find a feasible schedule with minimum build-ahead inventory. But if a feasible schedule is not possible, the causes of the infeasibility will be highlighted so the user can adjust capacity and/or due dates to correct the infeasibility.

The algorithm used in MRP-C is based on a conveyor model that characterizes the behavior at

each process in the system and it requires estimating two parameters, $l_p$ and $k_{pj}$. The minimum practical lead time is denoted by $l_p$ and it represents the processing time for a job with no queuing delays, but common (non-queuing) "dead" time should be included, such as the time that a job typically sits before being moved between machines. The unit capacity requirement, $k_{pj}$, which is actually the inverse of the practical production rate, represents the processing time for an item p on process j. These two parameters allow MRP-C to evaluate a simplified relationship between WIP and cycle time for an entire process.

The parameter $l_p$ will be equal to $k_{pj}$ for the case of processing a single part on a single machine. In the case of where the processing center is composed of several machines in series, then $k_{pj}$ will be the time required to process one item p on the slowest machine, i.e. the bottleneck production rate, and $l_p$ will be the sum of the time that is spent on each machine by item p. The last case is for batch production such as for a heat treating process where the processing time is independent of the batch size. For this case $l_p$ will be the batch processing time, the amount of time in the oven, the parameter $k_{pj}$ will be equivalent to the batch processing time $l_p$ divided by the total number of parts treated at once, the batch size.

MRP-C develops a production schedule in basically two phases. The first phase evaluates demand, WIP in the system, and the available capacity to determine if there is any infeasibility. After the infeasibilities have been addressed, the second phase starts at the furthest planning period and works backward in time to the current planning period to determine the minimum releases required to meet demand.

Tardif [93] further extended the MRP-C process to prioritize the production schedule based on the location of WIP within the manufacturing system. This process is composed of four steps. First, the algorithm determines the status of WIP at the preceding processing centers that are only one step upstream and feed the current stage. The algorithm then determines which requirements are covered and uncovered by the available WIP and schedules the covered requirements as early as possible. The missing capacity is then allocated to the uncovered requirements and the available capacity is then allocated to the uncovered requirements and scheduled as late as possible.

Tardif also included the ability to prioritize the scheduling with respect to the different capacity requirements when disaggregating into individual run quantities. The disaggregation scheme

orders items based on their capacity requirement and it is a generalization of the results from Gabbay [94] who developed an optimal solution algorithm for a production environment with serial product structure, no WIP, no yield losses, and where products have equal capacity requirements at each resource. The algorithm showed that the optimum solution occurs when the ordering of the products are in decreasing order of capacity requirements.

MRP-C has some advantages over MRP such as: the model it uses is more accurate and it explicitly considers capacity, and the planner is provided useful diagnostics. MRP-C also has some disadvantages. MRP-C relies on a heuristic, and thus it cannot be guaranteed that a feasible schedule will be found if one exists, but if a feasible schedule is found then it is truly a feasible schedule. Any errors produced by MRP-C will make the schedule more conservative, meaning that schedules will be more feasible than they need to be and should have a better chance of being successfully executed. The second problem is that it implies a push philosophy which makes it subject to all the possible drawbacks of push production although MRP-C is easily incorporated into a pull production method. In pull production a sequence is all that is required, but by incorporating MRP-C the actual "when" to produce a product is known. This can allow for the implementation of a pull system into a dynamic job shop environment to take advantage of the pull system's inherent knowledge of the operating conditions.

A pull system incorporated with MRP-C would be similar to a CONWIP system with MRP-C providing a production schedule for the system. The MRP-C generated release times will be close to the timing of the pull signals if all the parameters are correct for the MRP-C algorithms. Variability in the system will keep the times from matching exactly but on average the times should be consistent. If production falls behind, it can be made up by scheduling overtime or by adjusting the schedule at the next regeneration.

## Appendix IV: Common Liquid Packaging Issues

**Common Liquid Packaging Issues**

The process of packaging large quantities of a liquid into smaller containers or repacking from one container into multiple smaller containers has certain unavoidable issues that are present independent of what type of liquid is being packaged. Often these issues will make the transition from the typical batch and queue mass production system to a Just-In-Time production system very difficult. This section will discuss these issues and the implications on a transformation to a lean system.

**Cleanout Between Materials**

A changeover between materials in a traditional packing facility can be a very laborious task and if not done correctly can be very costly in the contamination of products, loss of material, employees sustaining injuries, or polluting the surrounding environment. The actual cleanout process can differ between packaging facilities but when viewed in the broader sense of the entire production system it is equivalent to a changeover in other manufacturing systems.

The cleanout process can cause problems when transitioning to lean if it is a time consuming process. The large downtime between materials will make production managers hesitant from moving from large batches to small batches with many rapid changeovers per shift, which is a goal of lean manufacturing. A rapid cleanout process is necessary prior to transitioning a packaging facility to a lean packaging system.

The typical packing station will have some length of piping that is shared by all material that is packaged at the station. The length of piping will differ in length from facility to facility. Some facilities with stationary tanks will use independent piping up to a manifold and only share the last few feet of piping and nozzles. Other facilities will use a shared pump and piping that runs tens or hundreds of feet from an outside loading dock for truck or rail tankers. Often facilities with significant lengths of piping also have the problem of many elevation changes in the piping and locations where material pools within the line making a thorough cleanout a very difficult and slow process. The poor layout of piping is often due to the need to rapidly increase packaging capacity without time to reengineer the facility or from the need to retrofit a facility to incorporate packaging processes into a facility that previously focused on other manufacturing processes. Unfortunately the solution to piping problems is often very costly and involves a complete redesign of the piping and a re-piping of the entire facility.

The actual method used to cleanout the piping will vary depending on the material being packaged and the facility packaging the material. Some chemicals such as glycerin, formaldehyde, or ammonium require cleaning using water while other chemicals such as ether or isopropyl alcohol only require a nitrogen and compressed air to cleanout the piping, but other materials such as petroleum products might require a solvent to clean the piping. The use of compressed air and nitrogen for cleanout can cause other issues to arise in addition to the downtime. This type of a cleanout can release the chemicals into the atmosphere which can be dangerous to the employees. During this research of packaging facilities, it was found that one facility performed pressurized nitrogen and air cleanouts of piping at the end of the day because workers could not be in the packaging room for a minimum of an hour from the start of the cleanout without protective clothing and a breathing apparatus. A cleanout at the end of the day implies that a station or line has a downtime from when the last container is filled until the next morning when a new material will begin to be packaged.

**Packaging Sequence and Material Grades**

Another issue of changeover between chemicals is the sequence in which some chemicals must be packaged. The sequencing of chemicals is important because certain chemicals have strong odors such as acetates or xylenes and other chemicals such as glycerin absorb odors, therefore glycerin should not be packaged after acetate has been packaged. Another issue of chemical sequencing deals with the type of cleanout used for the previous chemical and the susceptibility of the following chemical to absorb water. The following chemicals are an example of a few that should not follow a water cleanout: chloroform, dichloromethane, hexanes, xylenes, and toluene. Sequencing of chemicals is also important when a high purity level is desired because a chemical analysis of it will not show the contamination from the previous chemical. An example of this is that hexane and PET ether will appear pure even when a trace of one chemical is present in the lab work of the other chemical. The necessity to sequence materials in a packaging system adds another layer of complexity when trying to develop a pull production control system. The pull system must have an inherent intelligence that allows it to sequence the materials in the proper order to avoid the previously mentioned issues.

Packaging of various grades of the same chemical is sometimes a cause for changeovers to occur. One facility studied for this research is responsible for packaging three grades of products: the Lab grade of materials, Bio-Pharmaceutical grade of materials, and Micro-Electronic grade of materials, where each grade is composed of various sub-grades of materials. Numerous grades of

materials can be a difficult hurdle to overcome in transitioning to a lean system due to the introduction of a high variety of products. This hurdle becomes even greater when the production volume is low for materials but the materials are only available in high volumes.

**Transportation and Containment of Raw Materials**

The minimum order size of a truck tanker (typically 6,000 gallons) that some suppliers require is an issue to overcome in order to transition to a lean system. Ideally a packaging facility would be able to procure only enough material to meet the customer's need and perhaps a little extra for a small safety buffer of inventory. The difference between a customer order of 500 gallons and an incoming shipment of 6,000 gallons is significant because the large shipment will require much more time to package, create an over abundance of finished goods inventory (FGI), and tie up capital that could be used for other ventures. A lean system will function better if smaller more frequent shipments can be arranged.

A compartmentalized truck tanker, a normal tanker that is divided into three sealed compartments, can help to reduce the incoming shipment size, if the supplier is willing to supply a smaller order size. One drawback to the use of compartmental tankers is that there are federal regulations that restrict which chemicals can be transported on the same tanker, in order to protect the public if a mishap occurs. There is typically an increase in the cost of the chemical when the shipment size is reduced, which can be a drawback but can be offset by the packaging flexibility offered by a smaller shipment plus the financial savings due to a reduction in FGI. The last possible drawback to the use of compartment tankers is the supplier location and mix of chemicals to be purchased. A great distance between three chemicals that comprise one shipment can be a drawback to using a compartment tanker, such as if one supplier is in Virginia, another in Alabama, and the last is located in Texas. The supplier location can actually be beneficial if one supplier can be visited to purchase three compatible chemicals or if the three chemical suppliers are collocated. Collocated suppliers can be visited efficiently by the development of a "milk-run" strategy.

Compartment tankers are one possible solution to reduce shipment size, but the use of reusable container, perhaps 500 or 1,000 gallons in volume, is another solution if the supplier is willing to cooperate. Reusable containers can not only help to provide smaller more frequent deliveries of chemicals but also reduce changeover time due to the piping cleanout issues previously discussed.

These containers could be used directly at the packaging station with some reengineering of the layout of a packaging facility.

**Chemical Instability**

The type of container used to transport or hold some reactive chemicals must be chosen wisely because the container can rapidly deteriorate the quality of the chemical. The deterioration occurs as the chemical leaches some of the container material, which obviously worsens as the holding time increases. A liner can be used in a container to decrease the rate of leaching, but it can still contaminate the chemical if it is stored for a lengthy period of time.

Another form of chemical instability that leads to contamination or deterioration is the oxidation of some chemicals when exposed to the atmosphere. A tanker is typically vented to the atmosphere as it is being packaged or unloaded, which means that more air enters the tanker causing the last material to be removed to have been exposed to a great deal of air. The last of the material removed will be significantly lower in quality than the first material to be removed if the material deteriorates when exposed to atmosphere. Backfilling the tanker with nitrogen is often done to maintain the quality throughout the unloading process.

Evaporation of materials such as ether is another problem that often occurs during the warmer summer months at some packaging facilities. Hundreds of gallons can evaporate from a tanker during a period of a few days with high temperatures and when the tanker is in direct sunlight. The best way to solve this problem is to be able to unload the material quickly from the tanker to be placed in a sealed container.

**Various Packaging Containers**

Customer requirements increase the complexity for packaging facilities by requiring proprietary labels and distinct container sizes and shapes. An example of this occurred at one of the packaging facilities studied for this research, the company was purchased years ago by a conglomerate, but a single material that is packaged at the facility can leave the facility with two or three different labels, each labeled with a different company's name and format. The complexity of the labeling could easily be solved in a lean system by using postponement, but federal regulations exist requiring certain chemicals to be labeled immediately after packaging. A solution could be to use a generic label during the packaging process with the important information; chemical, lot number, packaging date and station, etc. A customer specific label

could be applied as the containers are being pulled from FGI and prepared for shipment. This would greatly decrease the amount of FGI required for each chemical, by a factor of two or three depending on the number of possible labels for each chemical.

The various sizes and shapes of containers is a more difficult problem to solve. Losing a customer by not offering the desired container is not an option, but perhaps through a good customer-supplier relationship certain concessions could be reached that would be mutually beneficial, such as a container that will meet the needs of many customers and reduce the possible variations for the packaging facility. In *The Machine That Changed the World*, Womack [95] discusses the different approach that Japanese manufacturers take toward their customers and suppliers. Suppliers and manufacturers come to mutually beneficial agreements on prices and delivery schedules where both parties can make a profit and both share information to help reduce costs which are split equally between the supplier and manufacturer. The manufacturer approaches the customer on his or her terms in order to "maximize the stream of income from a customer over the long term." The customers are made to feel as though they are part of an extended family and the manufacturer is the parent. This type of an approach develops trust and loyalty and more importantly a long-term relationship.

## References


1. Dockx, K., Y. De Boeck, and K. Meert, *Interactive scheduling in the chemical process industry.* Computers & Chemical Engineering, 1997. **21**(9): p. 925-945.
2. Guisinger, A. and B. Ghorashi, *Agile manufacturing practices in the specialty chemical industry - An overview of the trends and results of a specific case study.* International Journal of Operations & Production Management, 2004. **24**(5-6): p. 625-635.
3. Cormen, T.H., *Introduction to algorithms*. 2nd ed. 2001, Cambridge, Mass.: MIT Press. xxi, 1180.
4. Cho, F., L. Todd, K. Keafle, E. Grulke, M. Pittman and K. Saito. *Technical Discussion on Toyota Production System: Its Application and Human Elements*. in *Toyota Headquarter Office*. 2005. Nagoya, Japan.
5. Chase, C., J. Serrano, and P.J. Ramadge, *Periodicity and chaos from switched flow systems. Contrasting examples of discretely controlled continuous systems.* IEEE Transactions on Automatic Control, 1993. **38**(1): p. 70-83.
6. Perkins, J.R. and P.R. Kumar, *Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems.* IEEE Transactions on Automatic Control, 1989. **34**(2): p. 139-148.
7. Seidman, T.I. and L.E. Holloway. *Stability of a `signal kanban' manufacturing system*. 1997. Albuquerque, NM, USA: IEEE.
8. Seidman, T.I. and L.E. Holloway, *Stability of pull production control methods for systems with significant setups.* IEEE Transactions on Automatic Control, 2002. **47**(10): p. 1637-1647.
9. Johnson, S.M., *Optimal two- and three-stage production schedules with setup times included.* Naval Research Logistics Quarterly, 1954. **1**(1): p. 61-68.
10. Smith, W.E., *Various optimizers for single-stage production.* Naval Research Logistics Quarterly, 1956. **3**: p. 59-66.
11. Jackson, J.R., *Scheduling a production line to minimize maximum tardiness*, in *Management Science Research Projects*. 1955, University of California, Los Angeles, CA.
12. Yang, W.H. and C.J. Liao, *Survey of scheduling research involving setup times.* International Journal of Systems Science, 1999. **30**(2): p. 143-155.
13. Boysen, N., M. Fliedner, and A. Scholl, *Sequencing mixed-model assembly lines: Survey, classification and model critique.* European Journal of Operational Research, 2009. **192**(2): p. 349-373.
14. Allahverdi, A., J.N.D. Gupta, and T. Aldowaisan, *A review of scheduling research involving setup considerations.* Omega, 1999. **27**(2): p. 219-239.
15. Monden, Y., *Toyota production system : an integrated approach to just-in-time*. 1st ed. 1983, Norcross, Ga.: Industrial Engineering and Management Press. 423 p.
16. McMullen, P.R. *Multiple objective, mixed-model JIT assembly line sequencing with setups*. 1998. San Diego, CA, USA: Decis Sci Inst.
17. Burns, L.D. and C.F. Daganzo, *Assembly line job sequencing principles.* International Journal of Production Research, 1987. **25**(1): p. 71.
18. Bolat, A., M. Savsar, and M.A. Al-Fawzan, *Algorithms for real-time scheduling of jobs on mixed model assembly lines.* Computers & Operations Research, 1994. **21**(5): p. 487-498.
19. Kim, Y.K., C.J. Hyun, and Y. Kim, *Sequencing in mixed model assembly lines: A genetic algorithm approach.* Computers & Operations Research, 1996. **23**(12): p. 1131-1145.

20. Rahimi-Vahed, A. and A.H. Mirzaei, *A hybrid multi-objective shuffled frog-leaping algorithm for a mixed-model assembly line sequencing problem.* Computers & Industrial Engineering, 2007. **53**(4): p. 642-666.
21. Rahimi-Vahed, A.R., et al., *A multi-objective scatter search for a mixed-model assembly line sequencing problem.* Advanced Engineering Informatics, 2007. **21**(1): p. 85-99.
22. Tavakkoli-Moghaddam, R. and A.R. Rahimi-Vahed, *Multi-criteria sequencing problem for a mixed-model assembly line in a JIT production system.* Applied Mathematics and Computation, 2006. **181**(2): p. 1471-1481.
23. Rahimi-Vahed, A., et al., *Mixed-Model Assembly Line Sequencing Using Real Options*, in *Operations Research Proceedings 2006*. 2007. p. 161-167.
24. Rabbani, M., A. Rahimi-Vahed, and S. Torabi, *Real options approach for a mixed-model assembly line sequencing problem.* The International Journal of Advanced Manufacturing Technology, 2008. **37**(11): p. 1209-1219.
25. Miltenburg, J. and G. Sinnamon, *Scheduling Mixed-Model Multi-Level Just-in-Time Production Systems.* International Journal of Production Research, 1989. **27**(9): p. 1487-1509.
26. Kim, S. and B. Jeong, *Product sequencing problem in Mixed-Model Assembly Line to minimize unfinished works.* Computers & Industrial Engineering, 2007. **53**(2): p. 206-214.
27. Parrello, B.D., W.C. Kabat, and L. Wos, *Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem.* Journal of Automated Reasoning, 1986. **2**(1): p. 1-42.
28. Fliedner, M. and N. Boysen, *Solving the car sequencing problem via Branch & Bound.* European Journal of Operational Research, 2008. **191**(3): p. 1023-1042.
29. Kis, T., *On the complexity of the car sequencing problem.* Operations Research Letters, 2004. **32**(4): p. 331-335.
30. Morin, S., C. Gagné, and M. Gravel, *Ant colony optimization with a specialized pheromone trail for the car-sequencing problem.* European Journal of Operational Research, 2009. **197**(3): p. 1185-1191.
31. Solnon, C., et al., *The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem.* European Journal of Operational Research, 2008. **191**(3): p. 912-927.
32. Bolat, A., *Sequencing jobs on an automobile assembly line: objectives and procedures.* International Journal of Production Research, 1994. **32**(5): p. 1219.
33. Gagné, C., M. Gravel, and W.L. Price, *Solving real car sequencing problems with ant colony optimization.* European Journal of Operational Research, 2006. **174**(3): p. 1427-1448.
34. Inman, R.R. and D.M. Schmeling, *Algorithm for agile assembling-to-order in the automotive industry.* International Journal of Production Research, 2003. **41**(16): p. 3831 - 3848.
35. Lustig, I.J., and J.-F. Puget, *Program does not equal program: Constraint programming and its relationship to mathematical programming.* Interfaces, 2001. **31**(6): p. 29-55.
36. Ribeiro, C.C., et al., *A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints.* European Journal of Operational Research, 2008. **191**(3): p. 981-992.
37. Spieckermann, S., K. Gutenschwager, and S. Voβ, *A sequential ordering problem in automotive paint shops.* International Journal of Production Research, 2004. **42**(9): p. 1865 - 1878.
38. Miltenburg, J., *Level Schedules for Mixed-Model Assembly Lines in Just-in-Time Production Systems.* Management Science, 1989. **35**(2): p. 192-207.

39. Miltenburg, J. and G. Sinnamon, *Algorithms for Scheduling Multilevel Just-in-Time Production Systems.* Iie Transactions, 1992. **24**(2): p. 121-130.
40. Miltenburg, J. and G. Sinnamon, *Revisiting the Mixed-Model Multilevel Just-in-Time Scheduling Problem.* International Journal of Production Research, 1995. **33**(7): p. 2049-2052.
41. Ding, F.-Y. and L. Cheng, *An effective mixed-model assembly line sequencing heuristic for just-in-time production systems.* Journal of Operations Management, 1993. **11**(1): p. 45-50.
42. Aigbedo, H., *An assessment of the effect of mass customization on suppliers' inventory levels in a JIT supply chain.* European Journal of Operational Research, 2007. **181**(2): p. 704-715.
43. Aigbedo, H. and Y. Monden, *Simulation analysis for two-level sequence-scheduling for just-in-time (JIT) mixed-model assembly lines.* International Journal of Production Research, 1996. **34**(11): p. 3107-3124.
44. Aigbedo, H. and Y. Monden, *Parametric procedure for multicriterion sequence scheduling for Just-In-Time mixed-model assembly lines.* International Journal of Production Research, 1997. **35**(9): p. 2543-2564.
45. Kubiak, W., *Minimizing variation of production rates in just-in-time systems: A survey.* European Journal of Operational Research, 1993. **66**(3): p. 259-271.
46. Kubiak, W., *Balancing Mixed-Model Supply Chains*, in *Graph Theory and Combinatorial Optimization*. 2005. p. 159-189.
47. Kubiak, W., D. Rebaine, and C. Potts, *Optimality of HLF for scheduling divide-and-conquer UET task graphs on identical parallel processors.* Discrete Optimization, 2009. **6**(1): p. 79-91.
48. Kubiak, W., G. Steiner, and J.S. Yeomans, *Optimal level schedules for mixed-model, multi-level just-in-time assembly systems.* Annals of Operations Research, 1997(69): p. 241-241.
49. Steiner, G. and J.S. Yeomans, *Optimal level schedules in mixed-model, multi-level JIT assembly systems with pegging.* European Journal of Operational Research, 1996. **95**(1): p. 38-52.
50. Steiner, G. and S. Yeomans, *Level schedules for mixed-model, just-in-time processes.* Management Science, 1993. **39**(6): p. 728-735.
51. Sumichrast, R.T. and E.R. Clayton, *Evaluating sequences for fast paced, mixed-model assembly lines with JIT component fabrication.* International Journal of Production Research, 1996. **34**(11): p. 3125.
52. Sumichrast, R.T., K.A. Oxenrider, and E.R. Clayton, *An evolutionary algorithm for sequencing production on a paced assembly line.* Decision Sciences, 2000. **31**(1): p. 149-172.
53. Sumichrast, R.T. and R.S. Russell, *Evaluating mixed-model assembly line sequencing heuristics for just-in-time production systems.* Journal of Operations Management, 1990. **9**(3): p. 371-390.
54. McMullen, P.R., *JIT sequencing for mixed-model assembly lines with setups using Tabu Search.* Production Planning and Control, 1998. **9**(5): p. 504-510.
55. McMullen, P.R. and G.V. Frazier, *A simulated annealing approach to mixed-model sequencing with multiple objectives on a just-in-time line.* Iie Transactions, 2000. **32**(8): p. 679-686.
56. McMullen, P.R., P. Tarasewich, and G.V. Frazier, *Using genetic algorithms to solve the multi-product JIT sequencing problem with set-ups.* International Journal of Production Research, 2000. **38**(12): p. 2653-2670.

57. McMullen, P.R., *An ant colony optimization approach to addressing a JIT sequencing problem with multiple objectives.* Artificial Intelligence in Engineering, 2001. **15**(3): p. 309-317.
58. McMullen, P.R., *A Kohonen self-organizing map approach to addressing a multiple objective, mixed-model JIT sequencing problem.* International Journal of Production Economics, 2001. **72**(1): p. 59-71.
59. McMullen, P.R., *An efficient frontier approach to addressing JIT sequencing problems with setups via search heuristics.* Computers and Industrial Engineering, 2001. **41**(3): p. 335-353.
60. Mansouri, S.A., *A Multi-Objective Genetic Algorithm for mixed-model sequencing on JIT assembly lines.* European Journal of Operational Research, 2005. **167**(3): p. 696-716.
61. Mohammadi, G. and M. Ozbayrak, *Scheduling mixed-model final assembly lines in JIT manufacturing.* International Journal of Computer Integrated Manufacturing, 2006. **19**(4): p. 377 - 382.
62. Ponnambalam, S.G., P. Aravindan, and M. Subba Rao, *Genetic algorithms for sequencing problems in mixed model assembly lines.* Computers & Industrial Engineering, 2003. **45**(4): p. 669-690.
63. Ahmadi, R.H. and H. Matsuo, *A mini-line approach for pull production.* European Journal of Operational Research, 2000. **125**(2): p. 340-358.
64. Doganis, P. and H. Sarimveis, *Optimal scheduling in a yogurt production line based on mixed integer linear programming.* Journal of Food Engineering, 2007. **80**(2): p. 445-453.
65. Cho, K.-H. and J.-T. Lim, *On-line tracing supervisory control of discrete-event dynamic systems based on outlooking.* Automatica, 1999. **35**(10): p. 1725-1729.
66. Miltenburg, G.J. and T. Goldstein, *Developing Production Schedules Which Balance Part Usage and Smooth Production Loads for Just-in-Time Production Systems.* Naval Research Logistics, 1991. **38**(6): p. 893-910.
67. Leu, Y., P.Y. Huang, and R.S. Russell, *Using beam search techniques for sequencing mixed-model assembly lines.* Annals of Operations Research, 1997(70): p. 379-379.
68. Briant, O., D. Naddef, and G. Mounié, *Greedy approach and multi-criteria simulated annealing for the car sequencing problem.* European Journal of Operational Research, 2008. **191**(3): p. 993-1003.
69. Gupta, A.K. and A.I. Sivakumar, *Controlling delivery performance in semiconductor manufacturing using Look Ahead Batching.* International Journal of Production Research, 2007. **45**(3): p. 591-613.
70. Ramadge, P.J. and W.M. Wonham, *Supervisory Control of a Class of Discrete Event Processes.* SIAM Journal on Control and Optimization, 1987. **25**(1): p. 206-230.
71. Chung, S.L., S. Lafortune, and F. Lin, *Limited lookahead policies in supervisory control of discrete event systems.* Automatic Control, IEEE Transactions on, 1992. **37**(12): p. 1921-1935.
72. Chung, S.-L., S. Lafortune, and F. Lin, *Recursive computation of limited lookahead supervisory controls for discrete event systems.* Discrete Event Dynamic Systems, 1993. **3**(1): p. 71-100.
73. Kumar, R., H.M. Cheung, and S.I. Marcus, *Extension based Limited Lookahead Supervision of Discrete Event Systems.* Automatica, 1998. **34**(11): p. 1327-1344.
74. Takai, S., *Estimate based limited lookahead supervisory control for closed language specifications.* Automatica, 1997. **33**(9): p. 1739-1743.
75. Horn, C. and P.J. Ramadge. *Dynamics of switched arrival systems with thresholds*. 1993. San Antonio, TX, USA: IEEE.

76. Ushio, T., H. Ueda, and K. Hirai, *Controlling chaos in a switched arrival system.* Systems and Control Letters, 1995. **26**(5): p. 335-339.
77. Ueda, H., T. Ushio, and K. Hirai. *Oscillations in a single machine systems with limited continuous processing time policy*. in *International Conference on Nonlinear Theory and its Applications*. 1993. Hawaii, USA.
78. Li, W. and T. Ushio, *Control of a chaotic switched arrival system with controlled internal connections.* International Journal of Bifurcation and Chaos, 2006. **16**(3): p. 701-707.
79. Tian, Y.-P. *Detecting unstable periodic orbits in switched arrival systems*. 2003. Maui, HI, United states: Institute of Electrical and Electronics Engineers Inc.
80. Ushio, T., H. Ueda, and K. Hirai, *Control of Chaos in Switched Arrival Systems with N Buffers.* Electronics and Communications in Japan, Part III: Fundamental Electronic Science (English translation of Denshi Tsushin Gakkai Ronbunshi), 2000. **83**(8): p. 81-86.
81. Miltenburg, J., *Comparing JIT, MRP and TOC, and embedding TOC into MRP.* International Journal of Production Research, 1997. **35**(4): p. 1147 - 1169.
82. Miltenburg, J., *Balancing and scheduling mixed-model U-shaped production lines.* International Journal of Flexible Manufacturing Systems, 2002. **14**(2): p. 123-155.
83. Miltenburg, J., *Level schedules for mixed-model JIT production lines: characteristics of the largest instances that can be solved optimally.* International Journal of Production Research, 2007. **45**(16): p. 3555-3577.
84. Miltenburg, J., G. Steiner, and S. Yeomans, *A Dynamic-Programming Algorithm for Scheduling Mixed-Model, Just-in-Time Production Systems.* Mathematical and Computer Modelling, 1990. **13**(3): p. 57-66.
85. Holloway, L.E., *Modeling and Simulation of Manufacturing Systems Under Signal Kanban Policies*, in *2nd International Symposium on Scale Modeling*. 1997: Lexington, Kentucky.
86. Krieg, G.N. and H. Kuhn, *A decomposition method for multi-product kanban systems with setup times and lost sales.* IIE Transactions (Institute of Industrial Engineers), 2002. **34**(7): p. 613-625.
87. Askin, R.G. and J.B. Goldberg, *Design and analysis of lean production systems*. 2002, New York: Wiley. xiv, 533 p.
88. Hopp, W.J. and M.L. Spearman, *Factory physics : foundations of manufacturing management*. 2nd ed. 2001, Boston: Irwin/McGraw-Hill. xxii, 698 p.
89. Bartholdi, J.J. and L.K. Platzman, *Heuristics Based on Spacefilling Curves for Combinatorial Problems in Euclidean-Space.* Management Science, 1988. **34**(3): p. 291-305.
90. Vollmann, T.E., W.L. Berry, and D.C. Whybark, *Manufacturing planning and control systems*. 3rd ed. 1992, Homewood, IL: Irwin. xx, 844 p.
91. Monden, Y., *The Toyota management system : linking the seven key functional areas*. 1993, Cambridge, Mass.: Productivity Press. xxiii, 222 p.
92. Tardif, V., *Detecting Scheduling Infeasibilities in Multi-Stage, Finite Capacity, Production Environments*, in *Department of Industrial Engineering and Management Sciences*. 1995, Northwest University: Evanston.
93. Tardif, V. and M.L. Spearman, *Diagnostic scheduling in finite-capacity production environments.* Computers and Industrial Engineering, 1997. **32**(4): p. 867-878.
94. Gabbay, H., *MULTI-STAGE PRODUCTION PLANNING.* Management Science, 1979. **25**(11): p. 1138-1148.

95. Womack, J.P., D.T. Jones, and D. Roos, *The machine that changed the world : how Japan's secret weapon in the global auto wars will revolutionize western industry*. 1st HarperPerennial ed. 1991, New York, NY: HarperPerennial. viii, 323 p.

## Vita

John Thomas Henninger was born on September the 10th, 1976 in Murray, Kentucky.

Education:

- Murray State University
    - Attended from June 1994 until May of 1995.
- University of Kentucky
    - Bachelor of Science in Mechanical Engineering, May of 1998
    - Masters of Science in Mechanical Engineering, May of 1999

Honors:

- Outstanding Junior in Mechanical Engineering Department, 1997
- Outstanding Senior in Mechanical Engineering Department, 1998
- National Science Foundation Fellow, 1998-1999
- College of Engineering Staff Excellence Award, 2007

Professional Positions:

- Industrial Extension Engineer, Center for Manufacturing, University of Kentucky
    - August 1999 – present
- Part-time Lecturer, Mechanical Engineering Department, University of Kentucky
    - June 1999 – May 2000
- Design Engineer, ProTek Engineering, Lexington, KY
    - March 1998 – August 1999

Publications:

- J.T. Henninger and L. Holloway, "Stability Determination in a Class of Manufacturing Systems with Replenishment Signals," *Proceedings from 2010 American Control Conference*, Baltimore, MD, June 210. *Submitted September 21, 2009.*
- J.P. Medendorp, J.A. Fackler, T. Henninger, B. Dieter, and R.A. Lodder, "NIR spectrometry for the characterization of fuel components in a novel tamper-resistant pill bottle," *Journal of Pharmaceutical Innovation*, vol. 1, 54-61, 2006.