Performance Analysis of Manufacturing Systems Under Hybrid Control Policies

by

Asbjoern M. Bonvik

Sivilingeniør, Norwegian Institute of Technology (1988)

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

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

Author . . . .
Department of Electrical Engineering and Computer Science
May 3, 1996

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

Accepted by . . . .
Thomas L. Magnanti
Co-Director, Operations Research Center

Performance Analysis of Manufacturing Systems
Under Hybrid Control Policies

by
Asbjoern M. Bonvik

Submitted to the Department of Electrical Engineering and Computer Science
on May 3, 1996, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

We present a new class of control policies for manufacturing systems. These control policies are hybrids of existing policies that limit the local inventories (like kanban) and policies that provide a global flow of information (like basestock). Presently, kanban is widely used in practice, and is seen as the "state of the art" in lean manufacturing. We present several simulation studies, ranging in complexity from a simple tandem line to a semiconductor factory. These studies show that the hybrid policies achieve high service levels with significantly lower inventories than kanban. The hybrid policies are card-based, very simple to implement, and retain the organizational learning benefits of kanban.

We construct new performance analysis methods based on decomposition. This is used to analyze manufacturing systems controlled by hybrid policies. Novel elements of the decompositions developed in this thesis are that several different types of elementary components are used, and that both the flow of material and the flow of information are modelled. The resulting methods are very fast and reasonably accurate. This permits design of a factory and its operating policies simultaneously, without resorting to rules of thumb or time-consuming simulation studies.

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

Acknowledgments

This thesis was supported by many sources. Intellectually, the greatest source of support was my advisor, Stanley B. Gershwin. His never-ending supply of good ideas was a great inspiration. I also had the good fortune to work with Yves Dallery, who visited MIT in the summer of 1995. His ideas and helpful suggestions were instrumental to this work. In particular, the decomposition approach with a separate synchronization stage developed in chapters 4 - 8 of this thesis was suggested to me by him. My thesis committee members Donald E. Troxel and Stephen C. Graves also contributed with many valuable suggestions. The other students in Stan's group were a great support, and I could always try out ideas on them. The Group, thanks — you know who you are.

This work was supported financially in part by the Advanced Research Projects Agency under contract N00174-93-C-0035 and by the Norwegian Research Council under contract 100250/410.

Most of all, I want to thank my wife Jannicken and the kids, Vigdis and Thorbjørn, for helping me keep the whole thing in perspective.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is about the effects of various patterns of information flow in a factory. We are particularly concerned with the inventory effects of replacing decision making based on local inventory information with decisions based on global demand information, or combinations of the two types of information. To give an intuitive feel for what we are trying to achieve, we start with some analogies drawn from very different fields.

In Goldratt and Fox (1984) the impact of a bottleneck in a plant is explained through an analogy to a troop of boy scouts on an outing. One of the boy scouts is slower than the others, so a gap keeps opening between him and the scouts in front of him. This is then compared to the way inventory piles up in front of a bottleneck in a plant. However, Goldratt only told half the story.

As any ex-soldier will know, the main problem in a marching column is not a gap appearing in front of a slow marcher. The problem is variability amplification. If the first soldier in a column speeds up slightly, the second soldier will keep going at his old speed for a while. Then a small gap appears in front of him, indicating that he is falling behind. In response, that soldier also speeds up. But in order to close the gap, he needs to speed up a bit more than the first soldier did, and then slow down to the first soldier's speed when the gap is closed. Now, the third soldier sees a gap opening even faster than the second one did, so he has to speed up even more to catch up with the second soldier, and then abruptly slow down when this is done. This way, a small change in speed at the front of the column amplifies like a whiplash down the column. The last soldier in a column operating like this alternates between running flat out to catch up and marching in place to avoid bumping into those in front of him.

This is also a problem in a manufacturing system where each station in the chain relies on the status of local inventories to trigger production decisions. Human intervention tends to increase the problem, as is graphically demonstrated by the "Beer Game" played by students at the MIT Sloan School of Management. The students play roles of manufacturers, distributors, and retailers of beer. A small change in the demand for a particular brand of beer, imposed by the game management, tends to cause each successive link in the supply chain to over-react, and the game is awash in beer after a few iterations. This is amusingly described in Senge (1990).

There are traditional ways of coping with this variability amplification in a marching column. If the soldiers are green-horns, a very loud drill sergeant will be yelling out the cadence and forcefully admonishing the soldiers to close any gap that appears. With more experienced marchers, the column will most likely be singing a marching song. Any well-established infantry outfit has an inexhaustible supply of these. Both of these techniques has the effect of broadcasting the correct marching cadence to every soldier in the column simultaneously. The marchers are then less dependent on the local information they gather by watching the distance to the next soldier in order to regulate their speed, and the progress of the column as a whole becomes remarkably smoother.

There are analogies to this from other fields as well: When a column of cars slows down in dense traffic, each driver sees the brake lights of the car in front go on, the distance to that car diminish, and takes this as a signal to initiate braking. Someone that responds late will need to brake hard, which in turns requires those following to respond faster and brake even harder. There is an inherent risk of initiating a chain collision in this process. This risk was reduced some years ago by the introduction of high-mounted brake lights on cars. These have the effect of transmitting the information that a car is braking to other cars several positions down the column, without going through the reaction times of the drivers in between. The drivers farther back can then initiate soft braking at once, and the column comes to a halt more smoothly with less risk.

We are concerned with similar issues, like inventory levels and response times, and study how these are affected by factory control policies that are based on local inventory information, and by policies that are based on global broadcast of demand or production rate information. In our analogies, the decision makers at each stage still observe the local information and factor it into their decisions, even when global information is made available. We also look at control policies that combine these two types of information flow. We then construct very simple control schemes that utilize this information to achieve better performance than can be done by relying on either information source alone.

Our methods are an eclectic collection of the tools that could be brought to bear on this issue. We use discrete event simulations as the tool to use when all else fails. We also analyze mathematical models for some small systems exactly (under a certain set of assumptions), and develop fast, approximate methods for larger systems. We first survey the literature, then do some simulation studies of realistic production systems, before delving into the mathematics to develop faster tools.

1.1 Previous work

1.1.1 Control policies

There has been much work on individual operating disciplines, but relatively few comparison studies have been done. This may be partly because the different approaches have not been described within the same nomenclature. Buzacott and Shantikumar (1992) present a common framework to describe a wide class of distributed manufacturing scheduling policies governed by *production authorization cards*, or PACs for short. The policies we study are basestock, CONWIP, kanban, and variations thereof. These can all be seen as special cases of the framework of Buzacott and Shantikumar. The policies can be described by particular choices of the relevant control parameters and implemented by circulating kanban-like cards. Buzacott and Shantikumar (1993) report some performance analysis results for a set of systems similar to those we study, but with exponential service times and Poisson demand arrivals. Other similar control frameworks are the generalized kanban control system of Frein, Di Mascolo, and Dallery (1994) and the extended kanban control system of Dallery and Liberopoulos (1995).

The classical reference on basestock is Clark and Scarf (1960), where optimality is shown for a system with constant lead times and unlimited machine capacities. A paper by Kimball (1988) presents this policy in a more accessible form, and Lee and Zipkin (1992) develop an approximate performance analysis of this policy under the assumption of exponential service times and Poisson demand. One interesting observation on basestock is that a production stage controlled by this policy maintains a finished goods inventory ahead of demand. This is equivalent to the hedging point policies developed from an optimal control perspective by Kimemia and Gershwin (1983) and others, for systems with deterministic demand. These policies are optimal for a single unreliable machine in constant demand. See, for instance, Bielecki and Kumar (1988) and Gershwin (1994).

There is an immense kanban literature, and one survey paper is Berkley (1992). Descriptions of the Toyota Production System, of which kanban control is a part, can be found in Womack, Jones, and Roos (1990), Monden (1983), and Shingo (1989). In the technical literature, Tabe, Muramatsu, and Tanaka (1980) compare systems described as *push* and *pull*. The pull system is similar to the Toyota kanban system, and the push system seems to be an materials requirement planning (MRP) system driven by demand forecasts. MRP systems compute ordering and production quantities and times by analyzing the product's bill of material, and assume fixed lead times and unlimited capacities in the production system. The main performance measures used by Tabe, Muramatsu, and Tanaka are the amplification of ordering quantity and inventory level variance along the line. This is shown to be quite sensitive to forecasting errors. Kimura and Terada (1981) describe the Toyota kanban system in detail. They again compare this to the so-called push system, which is driven by unreliable demand forecasts, using variance amplification as a performance measure. They show that the batch sizes and production lead time of each stage have a major impact on the inventory fluctuations under kanban control, whereas the main factor in the push system again is the degree of forecasting error.

Optimizing the number of kanbans in a line has been a popular research topic, see, for instance Karmarkar (1986), Davis and Stubitz (1987), Bitran and Chang (1987), So and Pinault (1988), Mitra and Mitrani (1990), and Ohno, Nakashima, and Kojima (1995) for a sampling of the very different methods that have been applied. It seems that most kanban implementations set these parameters by rules of thumb or simple formulas. An example of such a formula is Toyota's $n \geq DL(1+\alpha)/a$, where n is the number of kanban, D is the demand rate, L the replenishment lead time, α a safety factor, and a the number of parts in a container (Sugimori, Kusunoki, Cho, and Uchikawa 1977). A less than satisfying aspect of this formula is that it is based on standard lead times, and thus does not reflect the lead time consequences of shop floor congestion and limited machine capacities. After the initial implementation, the kanban numbers are steadily decreased by reducing the safety factor as the system matures, as a part of the continuous improvement effort that is so important at Toyota.

Berkley (1991) shows that a common model of kanban systems is equivalent to a traditional tandem production line with finite buffers. This model assumes that kanban travel instantly to their destinations when they are detached from a part, and that both kanban and parts travel in quantities of one. If the line operates by *blocking before service* (also known as *information blocking* or *communications blocking*) the buffer size is the same as the number of kanban, and if it operates by *blocking after service* (alias *manufacturing blocking* or *production blocking*), the buffer size is one less. The usual assumption in the transfer line literature is that the last machine in the line is never blocked, which is not the case in a pull system. However, the transfer line literature should be applicable to kanban systems in saturating demand. Some useful papers are Buzacott and Hanifin (1978), Gershwin (1987a), Conway, Maxwell, McClain, and Thomas (1988), and Dallery and Gershwin (1992) as well as the treatments in the books by Gershwin (1994) and Buzacott and Shantikumar (1993).

Another kanban model, which we call *minimal blocking*, has been studied by several authors, starting with So and Pinault (1988). This model is the dominant kanban model in the literature, and is also referred to as having *dynamic buffers* to distinguish it from the tandem or fixed buffer or transfer line model. The difference from the tandem buffer model is that the minimal blocking model allows information to flow past a busy or failed machine as long as there are parts in its output buffer. Buzacott and Shantikumar (1992) describe a set of parameters for the production authorization card system that gives a kanban behavior. The resulting system is equivalent to the minimal blocking model. Mitra and Mitrani (1990) show that the minimal blocking model dominates the tandem buffer model in terms of output. Mitra and Mitrani (1991) extends this study to a line with Poisson demand, which is also studied by Di Mascolo, Frein, and Dallery (1996). So and Pinault state that the minimal blocking model "may be different from the operating concept of a typical pull system (e.g., see Kimura and Terada (1981))", but justify this to facilitate machine recovery from failures and to keep bottlenecks working even if there are failed machines downstream. We show that the hybrid policy we propose is a better way of achieving this result.

Spearman, Woodruff, and Hopp (1990) introduce the constant work-in-progress (CONWIP) policy. The main justification is to extend the scope of just-in-time manufacturing to systems where kanban is inappropriate. Kanban is only suitable for a high volume production environment with relatively few part types. It is not useful in an environment where expensive items with short lead times are rarely ordered, since it would require at least one of these items to be in inventory at all times. CONWIP instead limits the total inventory of all part types in the system and allows the part type mix and inventory location to vary as appropriate.

Van Ryzin (1987) and Van Ryzin, Lou, and Gershwin (1993) develop a control scheme that combines the basestock and kanban approach. They study a two-machine tandem line in constant demand by numerically solving an optimal control dynamic programming problem and observe that the optimal solution can be closely approximated by two boundaries: a hedging point for each machine and a finite buffer. This work has been followed up by Yan, Lou, Sethi, Gardel, and Deosthali (1994), where the two-boundary policies behave very well in a simulation study of semiconductor manufacturing. These policies are particularly attractive in systems subject to demand changes.

Veatch and Wein (1994) show that the basestock policies are never exactly optimal for a two-machine line with Poisson demand and exponential processing times. The difficulty with such policies is the possibility of accumulating large inventories that stay in the system for extended periods, because of capacity limitations. They also solve a dynamic program numerically and exhibit optimal policies similar to those found by Van Ryzin, Lou, and Gershwin. In the simulation experiments of Veatch and Wein, the best basestock policies come close to optimal. Depending on the parameters of the system, basestock policies may or may not be better than kanban policies. Kanban gives better results when the downstream machine is the bottleneck, while basestock gives better results when the bottleneck is upstream. An earlier simulation study of semiconductor manufacturing, Wein (1988), demonstrates that the release policy into a manufacturing system has a greater impact on the system performance than the dispatch policy followed within the system.

1.1.2 Performance analysis

This thesis uses discrete event simulations and decomposition methods applied to a continuous material approximation of the system for performance analysis. Exact analytic solutions to queuing networks are in general not available for any but the simplest systems. See Kleinrock (1975) for an introduction to queuing theoretic methods, and Buzacott and Shantikumar (1993) for applications of these methods to manufacturing systems.

Decomposition has been a quite successful method for fast approximate analysis of manufacturing systems. A starting point for the present work is the algorithm for synchronous manufacturing systems introduced by Gershwin (1987a). This was improved on by Dallery, David, and Xie (1988), and extended to a continuous material approximation to asynchronous systems by Dallery, David, and Xie (1989). The continuous material approximation was extended to arbitrary tree-structured production networks by Di Mascolo, David, and Dallery (1991), and Frein, Commault, and Dallery (1994) studied closed production systems where a finite population of pallets limit the amount of material in the system at any instant. The latter three papers all use an exact solution of a continuous material tandem line of two unreliable machines separated by a finite buffer as the building block. A description of the two-machine line and its solution can be found in Gershwin (1994).

In this thesis, we are concerned with *inhomogeneous* systems, where the machines can have different processing rates. Early work on these systems focused on transforming the system to one with equal processing rates. This approach was taken by Gershwin (1987b) by disaggregating each machine into two separate machines without a buffer between them. These machines would have very different rates of failures and repairs, and one would represent the limited processing rate, the other the failure behavior of the machine. All machines were then assigned processing rates equal to the fastest machine in the line. This method suffered from numerical problems when combined with the usual algorithms for solving systems of this type. Another variation is due to Dallery, David, and Xie (1989), where the line is homogenized by adjustments to the failure and repair rates of each machine. This worked well if the original processing rates were not too different. Another transformation proposed by Liu and Buzacott (1990) adjusts these rates in a different way, and achieves similar results.

For our purposes, this is not sufficient. We need to consider an external demand process that could be quite different in behavior from any of the machines in the line. For instance, one can imagine a situation where the demand is a slow, steady trickle while the machines in the production system are fast, but unreliable. On the other hand, one can also imagine a situation where a reliable production system with limited capacity needs to serve occasional intense bursts of demand. To be able to express these effects, we want a decomposition that works directly with the processing rates of the machines.

Several authors have proposed such decompositions. Choong and Gershwin (1987) derived a set of decomposition equations for lines with discrete parts, unreliable machines, and exponentially distributed operation times. This was improved upon by Gershwin (1989), where an extension of the algorithm by Dallery, David, and Xie (1988) was used to solve modified equations. For continuous material systems, one early method was developed by Glassey and Hong (1986). Their model does not account for reduced failure rates when machines are forced to slow down. Another method was developed by Semery (1987), who set the processing rates of each machine in the decomposition equal to that of the corresponding machine in the original line. Sometimes machines are forced to work at other production rates than their nominal rate due to the influence of other machines in the system, and a model incorporating this was developed by Alvarez Vargas, Dallery, and David (1991a). In this model, the machines in the decomposition are assigned a probabilistically weighted combination of the processing rates of adjacent machines. Convergence is sometimes a problem when using this model. Recently, a different decomposition of inhomogeneous lines was developed by Burman (1995). In this method, the failure and repair rates of the decomposed machines are determined from considering how starvations and blockages propagate in the line and how these are eventually cleared. Then the processing rates of the machines are determined from the relationship between flow rate, machine idle time, and conservation of flow in the line. This method is very fast and reliable.

There has been much work done in addition to the papers cited above. See Dallery and Gershwin (1992) for a comprehensive review and comparison, but few lines with other control disciplines than the one imposed by finite buffers in tandem have been analyzed. Some notable exceptions are the papers by Mitra and Mitrani (1990, 1991) where a minimal blocking control discipline is analyzed, and a paper by Lee and Zipkin (1992) analyzing the basestock discipline, both using decomposition methods. In his doctoral thesis, de Koster (1988) studied the behavior of several control policies, including basestock and CONWIP. His objective was to find the throughput of the different systems, and these control policies were approximated by tandem lines with suitably adjusted buffer sizes. These tandem lines were then analyzed by a method called *aggregation* (see Buzacott and Shantikumar (1993)). This gave good results for throughput, but poor estimates of inventory levels.

Chapter 2

Unified view of control policies

2.1 General model

To develop a common framework for the different control policies that have been proposed in the literature, we focus on the pattern of information flow generated by the control policy. This information can be transmitted by paper information carriers, by the material flowing through the system, by electronic means, or visually as empty spaces in buffers.

The information flow patterns can be divided into two classes: *global* information flows and *local* information flows. Global flows connect individual stages in the production system to the demand process, without going through intermediate production stages. Local information flows connect production stages to the adjacent buffer inventories. These two classes of information flow are illustrated in figure 2-1. Our graphical notation is to use circles for machines and triangles for buffers. This is in compliance with standards developed by the American Society of Mechanical Engineers (ASME) and Japanese industry standards (JIS-Z-8208), see also Arnell (1963), page 345. The machines drawn in heavy lines are those actually present in the system, and the one in light lines represents the demand process. Solid arrows represent material flows and dashed arrows represent information flows. For simplicity, we have not made any provisions in this figure for demand to wait for products if it cannot be satisfied from finished goods inventory, so this figure represents a lost sales model. Later figures do include demand waiting (backlog).

It is possible to make subdivisions within these two classes of information flows by specifying *when* the information is sent. For instance, a global information flow can be triggered when a demand event occurs, when the demand has been served, or even as an advance notice of a demand event that is expected to occur in the near future. This distinction is necessary when we consider systems where demand can be backlogged in sections 2.4, 2.5, and 2.6.

We use the graphical convention that dashed arrows originating from a buffer indicate messages passed when a part *leaves* the buffer, and similar arrows emanating from a machine indicate messages passed when a part leaves the machine, i.e., when it *enters* the buffer. Multiple arrows entering a machine, for instance one solid arrow (material) and one dashed arrow (information) indicate that one unit of each must be present before an operation is started. Multiple arrows leaving an object indicate material and/or information sent simultaneously along these paths. We will occasionally leave out some information buffers

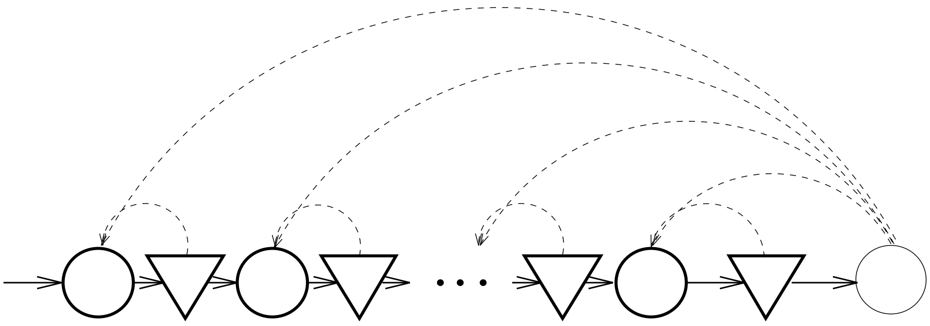

Figure 2-1: A unified model of control policies with lost sales

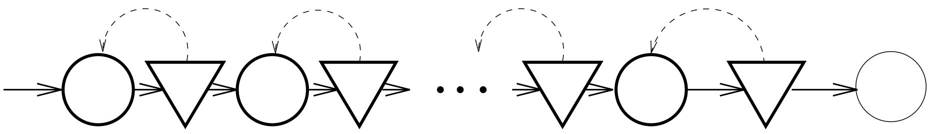

Figure 2-2: Information flow in kanban control

for clarity. For instance, in figure 2-1 the various messages may have to wait for the recipient machine to become available. This implies information buffers on each arrow representing an information flow.

We now look at some common policies to see how their information flows fit with this framework.

2.2 Kanban

Kanban control uses local information flows only, as shown in figure 2-2. It is often referred to as a "pull" policy, because each production stage in the chain only reacts to demands from the next downstream buffer. This treats the next downstream machine in the same way as the external demand process.

Kanban implementations take many forms. Berkley (1992) lists 24 different operational design factors that affect the behavior of a kanban system. These factors include the number of kanbans, the part container sizes, the material handling frequency, the material handling blocking mechanism, the part carrier capacity, and the distributions of such variables as setup and operation times. We consider a simple model where information and parts move instantly in batches of one, where the blocking model is blocking before service, and where there is only one part type. Under this blocking model, the machine does not load a new part from upstream when the downstream buffer is full, and remains empty until material is removed from the downstream buffer (Buzacott and Shantikumar 1993). A system operating under this control is equivalent to the tandem buffer systems of the transfer line literature (Berkley 1991).

Consider a particular kanban somewhere in the line. It circulates between a buffer and a machine. The number of kanbans in this loop limits the amount of material that can be

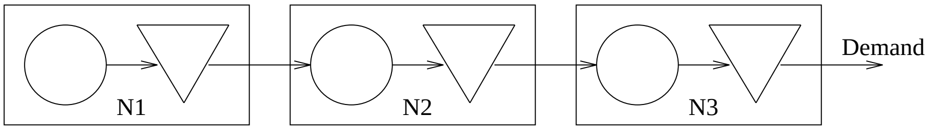

Figure 2-3: Inventory control cells in kanban control.

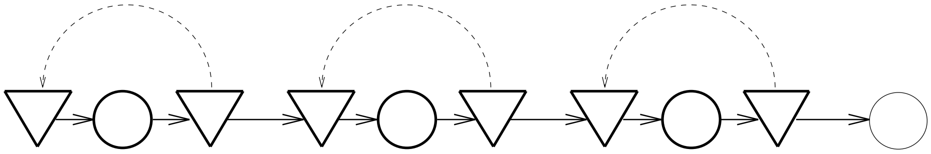

Figure 2-4: Information flow in minimal blocking control

in the *control cell* containing the machine and its downstream buffer. Note that the extent of the control cell is defined by the flow pattern followed by the kanbans. Figure 2-3 shows the control cells as rectangles, where N_i is the number of kanbans circulating in cell i. If a machine is down, it does not load material, and consequently no demand information is propagated upstream.

2.3 Minimal blocking

A variation of kanban control occurs if the information carriers are allowed to bypass a machine, even if this machine is busy or has failed. The resulting model is called *minimal blocking* (So and Pinault 1988). It still has local information flows as shown in figure 2-4. In the figure, we have split each buffer into two sections. The upstream section of each buffer is the output buffer of the machine upstream of the buffer. The downstream section is the input buffer of the downstream machine. When a kanban arrives at the buffer, it allows one part to be moved from the output section to the input section. A machine in this line can work if there is a part in its input buffer and the output buffer is not full.

Minimal blocking control limits the amount of inventory in cells consisting of a machine and its input and output buffers, as indicated in figure 2-5. To see the difference between the tandem buffer kanban control and the minimal blocking control, consider a k-machine line. Suppose that the output buffer of machine $k-1$ is full, that the input buffer of machine k is not empty, and the output buffer of machine k is not empty or full. Then machine k will try to do an operation. Suppose that it fails during this operation. Suppose also that a demand occurs during the failure. In the minimal blocking model, a part is delivered,

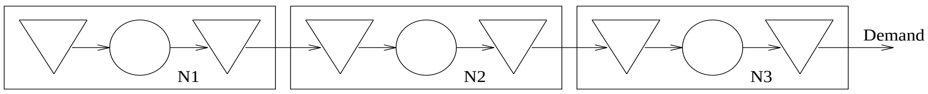

Figure 2-5: Inventory control cells in minimal blocking control.

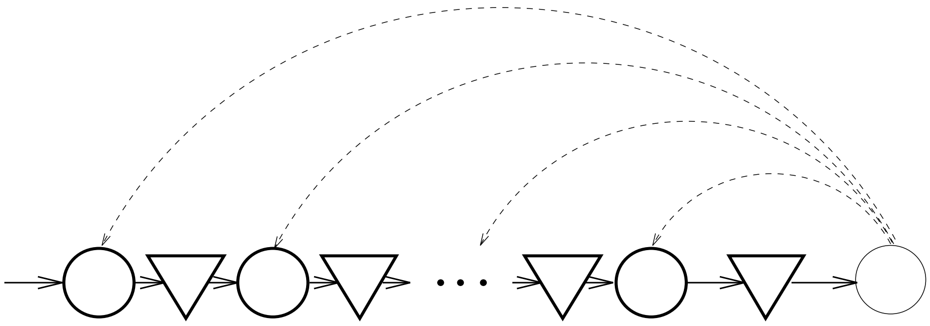

Figure 2-6: Information flow in basestock control without backlogging

a kanban is dispatched to buffer $k-1$, and a part is moved from the output to the input sections of the buffer. This causes machine $k-1$ to become unblocked and attempt a new operation. In our tandem buffer kanban model, this does not happen before machine k has been repaired, finished its operation and loaded another part. This behavior gives the minimal blocking model a higher throughput than the tandem buffer model for any given buffer allocation, but the consequences on WIP are not obvious.

Minimal blocking policies should not be confused with two-card kanban implementations. Toyota operates with a two-card implementation, where one class of card controls production and another class transportation. This simply controls the transportation resources in the same way as the machines, and does not imply a different blocking model.

It is interesting to note[1] that Toyota Motor Corporation uses a "push" type control with a global information flow for certain expensive or bulky items, such as seats, engines, and dashboards (Shingo 1989). When a job is scheduled for assembly, factory management can predict when this job will arrive at some operation in the assembly sequence (disregarding line stoppages, which are rare). The company can therefore initiate orders from suppliers so that these components are delivered to the assembly line in correct vehicle sequence, "just in time" for assembly. In fact, just-in-time production is not possible with kanban control alone, since a production system controlled by kanban only responds to the depletion of existing inventories. This distinction is worth noting, since kanban control is sometimes erroneously described as the same as just-in-time production or the Toyota production system.

2.4 Basestock

The basestock control is a Western approach to production and inventory control (Kimball 1988). It has global information flows only, as shown in figure 2-6. Whenever a demand occurs, a message is sent to every production stage, authorizing it to perform an operation.

Basestock control limits the amount of inventory between each production stage and the demand process as indicated in figure 2-7. Each machine tries to maintain a certain

[1]Thanks to Mitchell Burman for pointing this out.

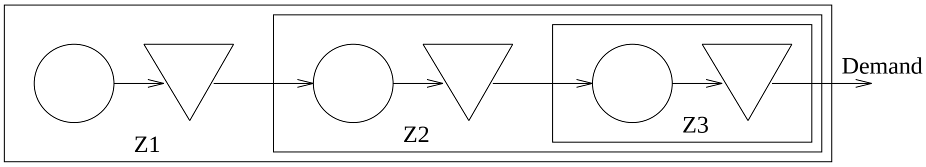

Figure 2-7: Inventory control cells in basestock control.

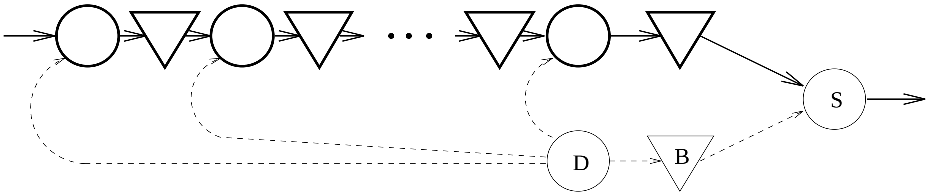

Figure 2-8: Information flow in basestock control with backlogging

amount of material in its output buffer, subtracting backlogged finished goods demand, if any. This amount is called the *basestock level* of the machine.

Basestock control does not impose limits on the individual buffer levels. This is useful in recovery from failures: When a machine fails, the demand process continues to remove material from the finished goods inventory, and the machines downstream of the failure operate normally until they become starved of parts to process. The machines upstream of the failed machine receive demand information directly from the demand process and operate as usual. There is therefore a build-up of inventory in front of the failed machine until it is repaired. When this occurs, the machine finds sufficient raw materials to catch up with demand in its input buffer without imposing an additional workload on the production system upstream of itself.

If backlogging is permitted, we must be more careful about when the information is passed, as indicated in figure 2-8. We have now constructed a more elaborate demand structure than the lost sales model in 2-1, consisting of the demand process D, a queue for waiting demand in the buffer B, and a fast and reliable synchronization station S. In this case, demand information is broadcast to all machines when the demand process D generates demand. Any demand that cannot be served immediately is backlogged in the buffer B before being instantly joined with available products by the synchronization station S. Note that the demand process produces information, the backlog buffer stores information, and the synchronization "machine" joins information and material.

The basestock policy is not a push policy in the sense described by Kimura and Terada (1981). Their "push" policy schedules production based on imperfect demand forecasts. The basestock policy we just described reacts to actual demand events, just like a kanban control does, and is therefore not subject to the problems caused by forecasting errors. But the basestock policy is not a "pull" policy, as described by Kimura and Terada, either, since the upstream production stages are not merely responding to the actions of the immediate downstream machine.

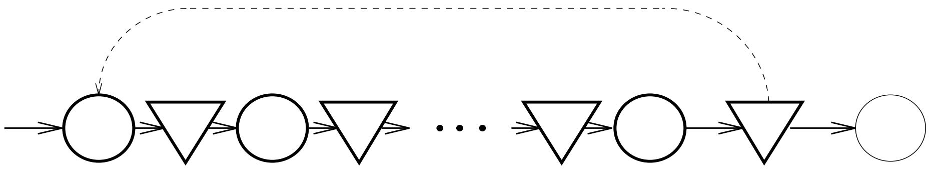

Figure 2-9: Information flow in CONWIP control without backlogging

The basestock control is closely related to the *hedging point* controls developed from optimal control formulations by Bai (1991), Gershwin (1994), and others. This control was studied by Bielecki and Kumar (1988) for a single unreliable machine, where it was demonstrated that holding zero inventory is optimal for a sufficiently reliable machine. The information flow is the same as in basestock control, but we refer to the resulting control as a basestock control if it responds to discrete demand events, and as a hedging point control if it tracks a target demand rate.

2.5 CONWIP

CONWIP control maintains a constant work in progress (Spearman, Woodruff, and Hopp 1990). When the preset WIP level is reached, no new job is authorized for release into the system until a job leaves. Demand that arrives when the system is full is usually kept on an external backlog. When a demand is served, a message is sent to the first stage in the line, authorizing the release of a new job to the system. The intermediate buffers inside the system can be considered infinitely large, or at least as large as the inventory limit for the system.

A CONWIP line can be seen as controlled by a single kanban inventory control cell encompassing all machines and all part types. This equivalency, which was noted by Spearman, Woodruff, and Hopp (1990), can be seen by considering the entire production system as a single machine with an output buffer. This aggregated machine starts another job whenever a piece is taken out of its output buffer, just as in kanban control.

If backlogging is permitted, we must again specify when information is passed. In figure 2-10, information is sent to the first machine whenever the synchronization station S has joined a demand with a product. The most important difference from basestock is in the timing of the information flow: In basestock, machines are notified when a demand arrives, but in CONWIP, the notification takes place when a demand is served. When backlogging is permitted, a demand can be stored in the backlog buffer for some time before it is served. This causes the information to the first machine to be sent later in CONWIP than in basestock control.

The main effect of this difference is in failure recovery. If a machine fails in a CONWIP line, the material downstream of it is eventually removed from the system by the demand process. These demand events cause the release of new jobs to the system. If the machine stays down long enough, these jobs and the jobs already in the system upstream of the failed machine accumulate in the buffer immediately upstream of this machine. CONWIP control stops admitting new jobs to the system when the inventory limit is reached, while basestock

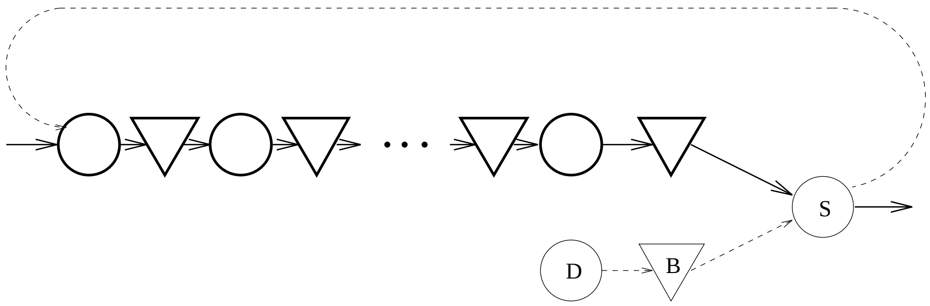

Figure 2-10: Information flow in CONWIP control with backlogging

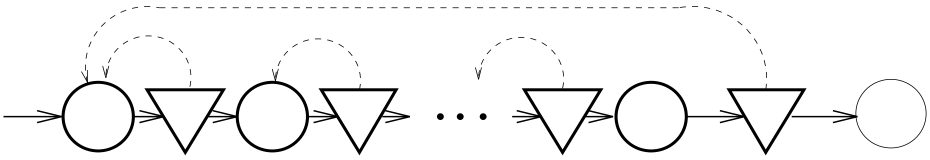

Figure 2-11: Information flow in CONWIP/kanban hybrid control.

control allows an unlimited number of jobs to accumulate in front of a failed machine.

2.6 Hybrid policies

It is not hard to visualize combinations of the information flows just described. For instance, we can imagine a system controlled by a single CONWIP-type global information flow acting as a release control combined with finite buffers between production stages limiting local inventory build-ups. This is illustrated in figures 2-11 an 2-12.

We can also imagine other hybrids, such as basestock and minimal blocking, which corresponds to the extended kanban control developed by Dallery and Liberopoulos (1995). We investigate these policies further in the following chapters, establish which policies perform better in various environments, and identify which features of the policies contribute to superior performance.

In fact, we show that the hybrid policies outperform the pure policies they are based on. Intuitively, this is because the hybrids contain two (or more) pure policies as special cases, so optimizing their parameters will be an optimization over a larger space than in the pure

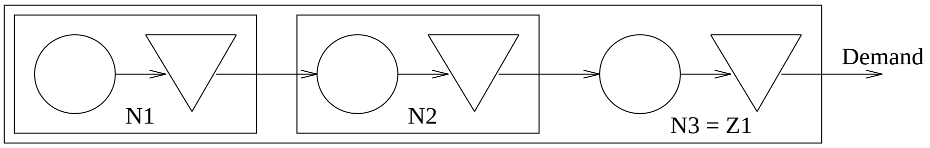

Figure 2-12: Inventory control cells in CONWIP/kanban hybrid control.

policies. It follows that the best parameter choice for a hybrid policy will perform at least as well as the best of the contained pure policies with optimal parameter choices.

Chapter 3

Simulation studies

In this chapter, we compare the control policies described in chapter 2 through simulation studies of realistic production systems. The advantage of simulation is that we can build models that represent arbitrary systems and control policies without introducing approximation errors. The disadvantage is the time required to build and run the resulting model. In later chapters, we derive faster, approximate methods for analyzing some of the control policies.

3.1 A case from Toyota Motor Company

3.1.1 Purpose

Kanban means "card" in Japanese, and refers to a shop floor control policy synchronizing manufacturing processes that make parts with those processes that consume the parts. This section presents results from a study of a typical component production line in Toyota Motor Company, with the objective of identifying better operating principles than kanban. Despite its significant success, kanban control is not flawless. Two concerns were mentioned by our contacts in Toyota: The amount of inventory held during normal operations, with the occasional machine failure; and the response of the line to changes in the product mix or demand rate.

3.1.2 Model

We focus on a model of a typical four-machine line that makes components that go into final assembly at the plant in Nagoya, Japan. This plant produces the Toyota Supra, among other models. The line is currently operated by kanban control.

We approach this matter through a series of simulation experiments. The model used was written for the purpose in the C programming language. It uses the UNIX system function *random* to generate uniformly distributed pseudorandom numbers, and converts them to the needed distributions by methods described in Rubinstein (1981). To supply the computing power needed for our experiments, we distributed the simulations on a network of twelve Sun workstations.

The model was validated by comparing its output to results derived from numerical solutions of Markov chain formulations of the same systems. We did this for one-, two-, and three-stage lines with exponential failure, repair, and operation time distributions. The lines were controlled by kanban, basestock, and hybrid policies (as described in chapter 2) in Poisson and saturating demand. These systems are solvable by analytic methods. For each of these reference systems, we did a series of experiments where simulation results with tight 95% confidence intervals were obtained through a large number (50-100) of replications of the same data, each with a different random number seed. Each replication in this validation series was run for 120,000 units of simulated time, with an additional warm-up period of 9,600 time units before any data logging started. The confidence intervals were estimated by large sample methods described in Arnold (1990).

We make some common assumptions across all scheduling policies:

- The system makes a single part type.
- Material is transported in units of one without delay.
- Information, such as kanban, is transmitted instantly.
- Machines operate asynchronously, so an operation can start whenever a part is present and the proper authorization has been received.
- Parts follow a first come, first serve dispatch policy at all machines.
- The time to fail measures working time, not clock time. This failure model is known as *operation dependent failures* (Buzacott and Hanifin 1978), and implies that no machine can fail unless it is working on a part.
- Any demand that cannot be satisfied from finished goods inventory is lost to the system. This reflects an assembly line stockout, where the entire line stops until parts arrive. In a Toyota plant, stoppages are compensated for by working longer hours until the production plan for the day is met. One can therefore say that the backlog information is maintained elsewhere in the factory.

There are several reasons for making these assumptions: One is to simplify our study by reducing the number of variables considered. Another reason is that several of the variables we left out, such as kanban transmittal time and batch size, are implementation details rather than essential aspects of the different control policies. By creating an ideal implementation, we illuminate the inherent behavior and limitations of the control policies. Finally, we have adapted the control policies to the system we study, and have made a non-standard assumption about the demand backlog behavior. This assumption is not essential to our control policies; they are easily adapted to different production systems.

3.1.3 Simulation results

Performance measures

One very important performance measure in the factory is the *service level* (or fill rate) at the assembly line. This is the fraction of all demands that find a component ready for use when the demand occurs.

Another important performance measure is the amount of *inventory* in the system. We define the inventory as the amount of material that has been loaded on the first machine, but has not yet been delivered to satisfy demand. We do not consider parts that are authorized for loading at the first machine inventory until they are actually loaded. In the following, we study the relationships between service level and inventory for the different control policies. We do this both for a constant demand rate, and for a case where the demand rate changes over time.

Some influential previous papers, e.g., Tabe, Muramatsu, and Tanaka (1980), use the *variability amplification* of the production ordering quantities along the line as their main performance measure. They defined this to be the ratio of order size variance at stage i divided by that of stage $i-1$. Measures like these are important in deciding how long production lines to use when designing a factory, and also indicate what load the system imposes on upstream suppliers. Our work assumes fixed ordering quantities of one, but does not fix the reordering intervals. That is, one unit of material is loaded into the machine whenever the machine is authorized to do this and material is available. Thus, a related performance measure is the variability of the intervals between successive loading events at each machine.

Constant demand rate

Our starting point is a four-machine line feeding parts to the assembly line. The assembly line is modeled as a deterministic demand process pulling one part per unit of time out of the finished goods buffer. If there is no part, the assembly line stops, and does not restart before parts have been made. The production rate is then equal to the fraction of requests from the line that were satisfied, i.e., the service level.

The line we simulated has operation times with mean 0.98 minutes and a standard deviation of 0.02 minutes. We modelled this by a lognormal distribution with parameters -0.02 and 0.02. (A random variable X is said to be lognormally distributed with parameters μ and σ if a random variable Y is normally distributed with mean μ and standard deviation σ, and $X = \exp(Y)$. This gives a mean $E[x] = \exp(\mu + 0.5\sigma^2)$ and variance $\sigma_x^2 = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1)$ (Rubinstein 1981; Drake 1967).) We chose the lognormal distribution because it can be sampled very efficiently and can represent low-variance distributions without numerical instabilities. The failure and repair distributions are assumed to be exponential with mean time to fail 1,000 minutes and mean time to repair 3 minutes. This gives each machine an isolated production rate of 1.01695. Even if machine failures are rare in this system, they create an important source of variability when the utilization is this high.

If our system did not have any variability, it would operate at a perfect service level of 1.0 with an inventory of 4 machines $*$ .98 minutes operation time $*$ production rate 1.0 parts per minute = 3.92 parts. In this case, material is only in the system when it is being worked on by a machine. We expect different control policies to have different relationships between inventory and service level in realistic systems with variability.

Our first experiment was to enumerate and simulate all kanban configurations with buffer space less than 5 for buffers 1, 2, and 3, and with buffer space less than 10 for finished goods. This gave 1,250 cases. The reasoning for choosing a larger last buffer is that inventory is not available to satisfy demand unless it is in that location, so we want to keep most of the WIP there.

We did the same experiment for the minimal blocking policy, for another 1,250 cases. Next we ran the same line under CONWIP control, trying all inventory limits in the range 1 to 25. The upper limit corresponds to the total buffer space in the largest kanban configuration we tried.

Another plausible choice is basestock control where the basestock levels reflect the time spent in each production stage times the production rate. For our system, this approach gives basestock levels of 1, since each part spends .98 time units in each stage on the average and we aim for a production rate of 1.00. Each part will then on the average spend just .02 units of time in each buffer before entering the next stage. One may surmise that the basestock control will lead to a smoother flow of material than CONWIP, but perhaps at increased inventory levels. We tried this scheme with the same range of overall inventory limits as in CONWIP.

Finally, since we know that basestock and CONWIP can accumulate excessive inventory, we also tried the hybrid policy with limits on total inventory and individual buffer levels. We again enumerated all buffer configurations in the range 1 through 5 for internal buffers, but let the finished goods buffer take on values up to 25. This gave another 3,000 cases to be simulated.

Each case in this series was run for 240,000 minutes of simulated time with an initial warm-up period of 9,600 minutes. Each one was started from a different random number seed.

Inventory results

In figure 3-1, we have plotted inventory against service level for kanban and hybrid control. Good parameter choices are those that attain high service levels at low inventories, so for each simulated case, we noted this pair of performance measures. The curves in the figure are the convex hulls of all service level/inventory pairs obtained for the kanban and hybrid policies. That is, each point on these curves corresponds to the parameter choice within a particular policy that achieved that service level with the least inventory. All the other parameter combinations gave data points inside the convex hull, and therefore gave less efficient usage of the inventory. The preferred policies at all service levels had a relatively large finished goods buffer and small internal buffers.

The hybrid policy makes a better service level/inventory compromise than kanban at all service levels. The vertical distance between the curves in figure 3-1 is the difference in inventory required to attain a particular service level. This difference becomes larger as the required service level increases. This graph is an important result: It says that a small change in the flow of information can produce a substantial reduction in inventories compared to kanban control. Kanban control is currently considered the state of the art in lean manufacturing.

The other control policies we tested generally fall between the kanban and hybrid curves in this plot, but sometimes exceed the inventory level of kanban at the same service level. None of the policies have lower inventories than the hybrid policy at any service level.

To investigate this further, suppose we get the following objective: *Find a configuration*

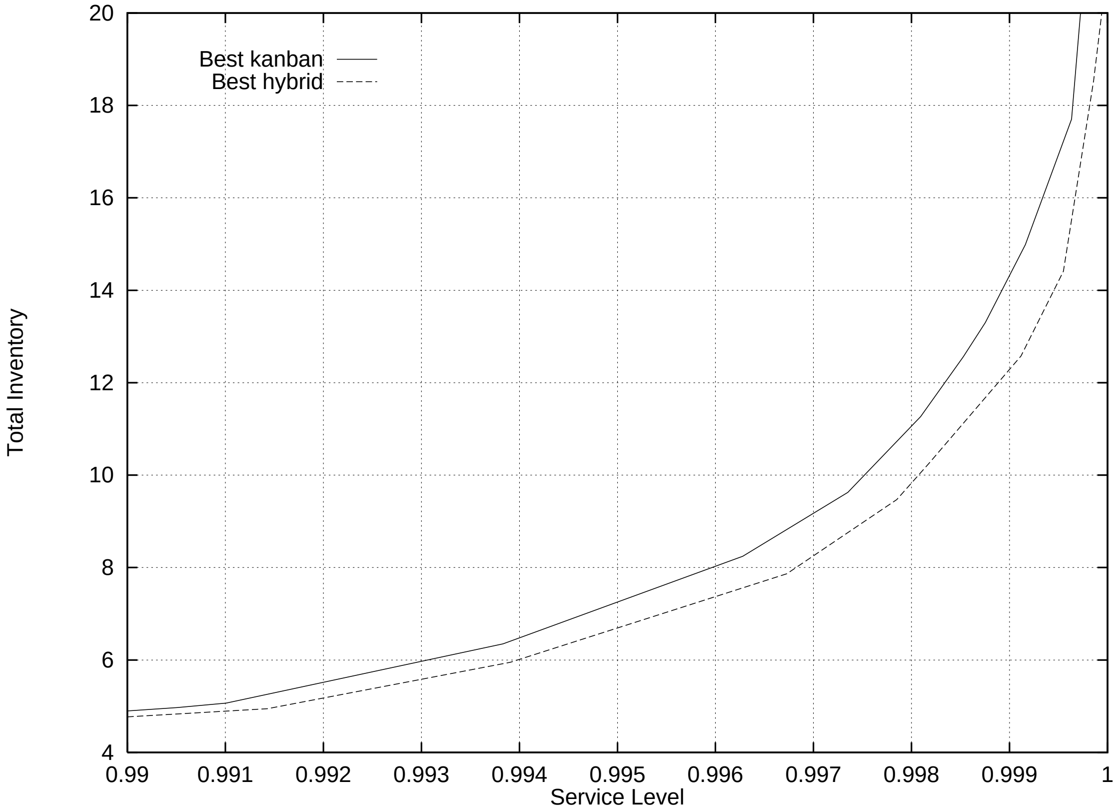

Figure 3-1: The tradeoff between service level and inventory

Policy	Description	Buffer sizes				Basestock levels			
Kanban	Finite buffers, blocking before service	2	2	4	10	∞	∞	∞	∞
Min. block.	Movable buffers	2	2	4	9	∞	∞	∞	∞
Basestock	Buffer target levels, infinite buffers	∞	∞	∞	∞	1	1	1	12
CONWIP	Single inventory limit, infinite buffers	∞	∞	∞	∞	0	0	0	15
Hybrid	Single inventory limit, finite buffers	2	3	5	15	0	0	0	15

Table 3.1: Best configurations that achieve the stated objective

Policy	Service level	Inventory
Kanban	0.99916 $\pm$ .00006	15.82 $\pm$ .05
Minimal blocking	0.99909 $\pm$ .00005	15.62 $\pm$ .05
Basestock	0.99918 $\pm$ .00006	14.60 $\pm$ .02
CONWIP	0.99922 $\pm$ .00005	14.59 $\pm$ .02
Hybrid	0.99907 $\pm$ .00007	13.93 $\pm$ .03

Table 3.2: Performance of best configurations

that gives a 99.9% service level using the least possible total inventory. This represents the task of finding the best way, by modifying information flow and control parameters, to achieve a management-set service target. We selected the data points from the previous results that came close to this service level, and did 30 replications with different random number seeds of each one. The run length was again 240,000 minutes with a 9,600 minute warm-up period for each replication.

We found the solutions shown in table 3.1, choosing configurations so that the entire 95% confidence interval for service level was above .999. For the kanban, minimal blocking, and hybrid policies, multiple configurations achieved the service level target with insignificant differences in inventories. Note that under our assumptions, kanban and CONWIP control become special cases of the hybrid control, while the minimal blocking control has a different blocking model. The only difference between basestock and CONWIP in this table is that we constrained the basestock levels to be non-zero in basestock control, but constrained them to be zero for the internal buffers in CONWIP control. Again, we emphasize that the apparent close relationship between CONWIP and basestock is in part due to our assumption of lost demand. Table 3.2 shows the performance of the best configurations found for each policy. The 95% confidence intervals were found by the formula

$$\overline{X} \pm 1.960 S/\sqrt{N}$$

where $\overline{X}$ is the sample mean, S the sample standard deviation, and N the sample size. This assumes that the sample is "large", but does not assume any underlying distribution of the sample points (Arnold 1990).

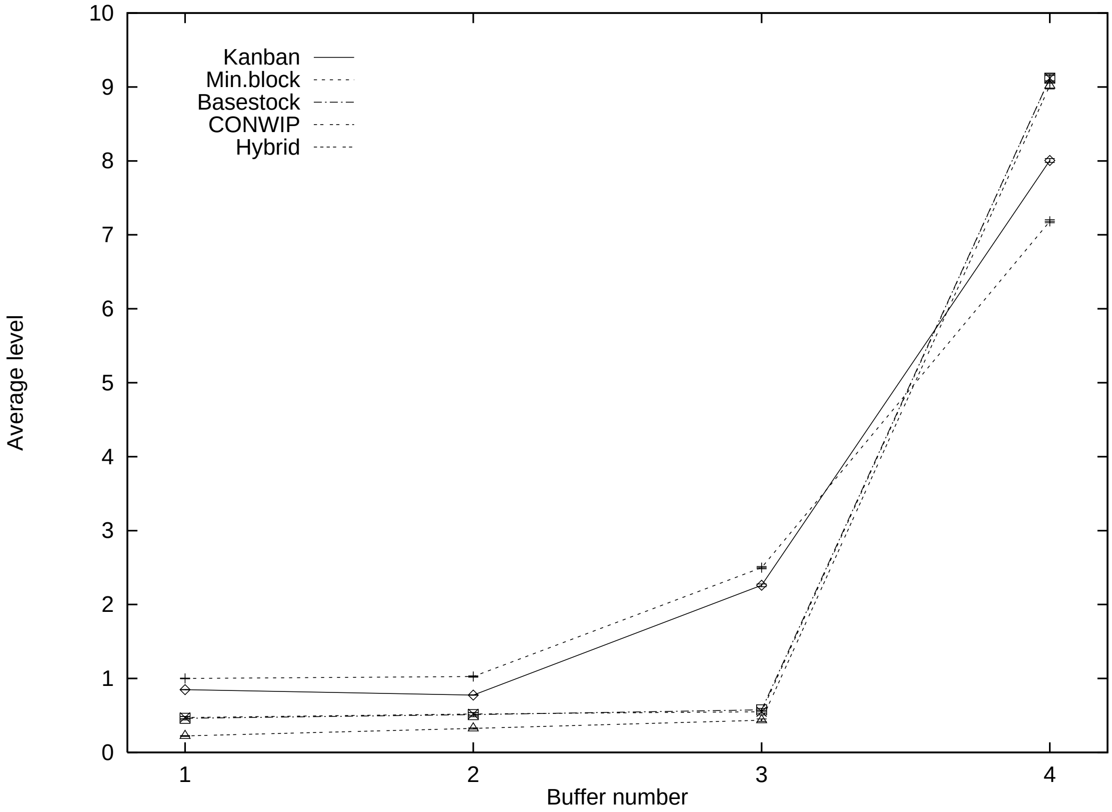

Figure 3-2: Location of inventory

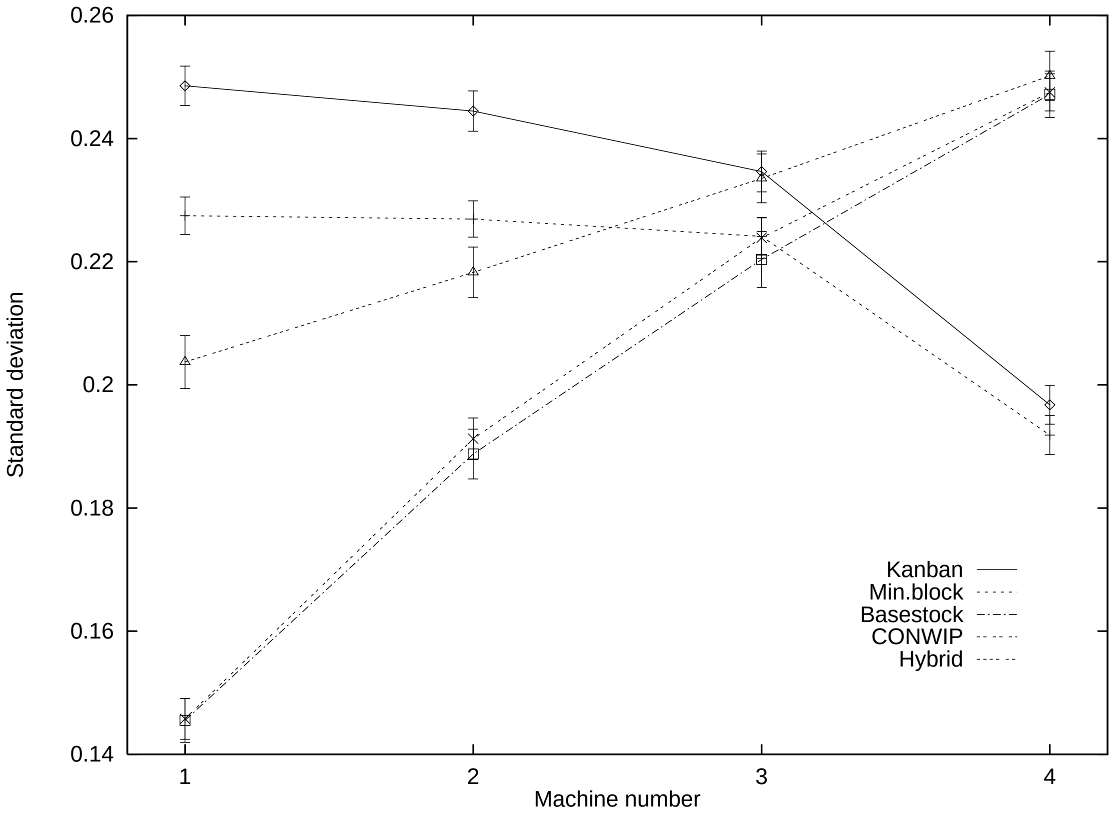

Figure 3-3: Variability of times between loading events

In this case, changing from the best kanban configuration to the best hybrid configuration produced an inventory reduction of 12% in an already lean system. As the curves in figure 3-1 demonstrated, this advantage increases to near 20% if the service level target is at 99.95%, but is smaller if the ambitions are lower. Another important observation that can be made from these simulations is that the average inventory locations of the policies are distinctly different. This is shown in figure 3-2, where we have plotted the average level of each buffer under the different control policies. Kanban maintains more inventory internal to the line than the other policies, because the kanban policy is explicitly to fill up the buffer whenever possible. This behavior is even more pronounced in the minimal blocking model than in the tandem buffer kanban model. The best basestock and CONWIP policies for this system have almost identical distributions of material along the line.

Variability amplification

We now turn to the amplification of variability in the line. Figure 3-3 shows the standard deviation of the times between loading events at each machine for the different control policies. These results are taken from the same series of experiments as in the previous section, and the configurations of table 3.1 were used.

Starting from machine 4, it is evident that kanban amplifies the variability as one proceeds upstream. The minimal blocking model has less variability, but about the same amplification as the tandem buffer model.

In contrast, the other policies appear to dampen the variability, although they start from a higher level of stage 4 variability. This is because a kanban controlled machine tries to track the immediate downstream demand process, while adding its own noise to the demand process seen by the next upstream machine. The other policies send demand information directly to the first machine, so the process of introducing material to the line is much less noisy. The variability of the machines then add noise as the parts move downstream. The basestock policy gives a slightly smoother flow than CONWIP, with the largest difference in the middle of the line, but the difference does not seem to be significant.

In summary, our versions of basestock, CONWIP, and hybrid control all vastly outperform kanban by the same performance yardstick that was used by Tabe, Muramatsu, and Tanaka as well as Kimura and Terada to argue the merits of pull control over push.

3.1.4 Changing demand rate

Since this production line is quite reliable, a more important consideration than the response to machine failures is the response to adjustments in the demand rate. Such adjustments are done occasionally at Toyota to reflect changes in market conditions. Our experiment was to run the line for 2,400 minutes at a rate of 1 minute between jobs, then increase the time between jobs to 1.25 minutes for another period of 2,400 minutes, and go back to the 1 minute cycle for the last 2,400 minutes of the experiment. This run was prefixed by a warm-up period of 1,200 minutes, where the line was running at 1 job per minute. We extracted the buffer level trajectories for each buffer and for total inventory as a function of time for each run of the experiment. Then we averaged these trajectories over 50 runs, each one with a different seed to the pseudo-random number generator. The resulting data were aggregated by averaging over time windows of 60 minutes to remove high frequency fluctuations that were irrelevant to our study. We did this experiment with the best kanban, CONWIP, and hybrid configuration from the previous section without any adjustment to the parameters to reflect the changed demand rate.

Figure 3-4 shows the results. The topmost three lines are the total inventory levels (WIP) for kanban (using the tandem buffer model), CONWIP, and hybrid as functions of time. Kanban has a higher total inventory at all times, but the difference grows much larger in the period of reduced demand. We note that the inventory levels are more variable in the periods of high demand than in the periods of low demand, due to less excess capacity to recover from random fluctuations. It also appears that kanban has more variable inventory levels than the other policies.

The next group of three lines shows the finished goods inventory (FG) over time for the three policies. For this group, the order is reversed from the total inventory case. Kanban has less finished goods inventory than the other policies, and its level increases less sharply when the demand decreases.

The last group of lines show the level of buffer 3 (B3). Note that kanban maintains a much higher inventory in this internal buffer than the other two policies. When the demand is decreased, CONWIP and hybrid control reduce the inventory in the internal buffers almost to zero, but kanban *increases* its internal buffer levels. The behavior of buffers 1 and 2 are similar to that of buffer 3, but at lower levels, as we have already shown

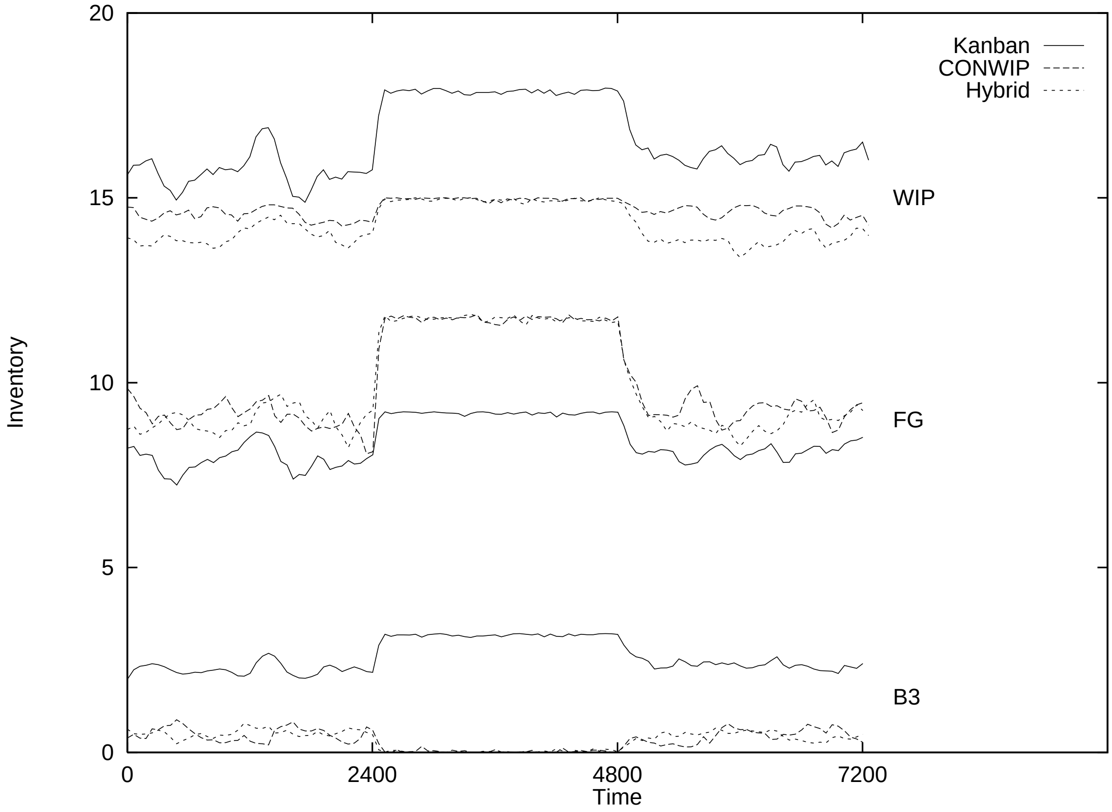

Figure 3-4: Inventory trajectories with changing demand

in figure 3-2.

3.1.5 Discussion

A kanban line ensures the overall production rate by making a few parts ahead of time and storing them in the buffers. These parts can then be used to let downstream machines continue working even if there is a disruption *upstream*. The kanban numbers regulate the size of these buffers, and the production rate is regulated by buffers filling up and blocking production. The hybrid policy, on the other hand, regulates the production rate by holding the release rate to the system equal to the demand rate. This keeps the internal buffers mostly empty. The empty space is then used to let upstream machines continue working, filling up this space, even if there is a disruption *downstream*. This achieves the same boost in production rate by partially decoupling the machines as if parts were stockpiled, but without the inventory cost.

Another important reason for the performance gap between these policies is the difference in information distribution. In kanban control, information flows only when material is moved. This is somewhat relaxed in the minimal blocking policy, but the information flow is still interrupted by the first empty buffer. The other policies distribute demand information directly to all machines whenever demands occur. This information passing is explicit in basestock control and implicit in CONWIP and hybrid control, where the demand information is sent to the first machine only. Decoupling the information flow from the part movements gives added stability against disruptions. This becomes increasingly important as the utilization approaches capacity. In our level schedule experiments, the effect of the hybrid policy as shown in figure 3-1 can be compared to entirely eliminating operation time variability or doubling machine reliability. In contrast to these potentially expensive ways of improving the system, changing the movement of kanban achieves the same benefit at essentially no cost.

In this study, there is only a small difference between the two most common models of a kanban system, represented by the tandem buffer kanban policy and the minimal blocking policy. The minimal blocking policy has higher service level and higher inventories than the fixed buffer kanban policy for the same buffer sizes. For other systems, the difference could be larger, but the service level advantage of the minimal blocking policy might not be enough to compensate for the increased inventories. For instance, if we set buffer sizes of 2, 2, 4, and 10 along the line, the tandem buffer kanban policy has an average inventory of 15.82 and the minimal blocking policy 16.53, a difference of 4.5% in favor of tandem buffers. At the same time, the tandem buffer policy has a service level of .99916 and the minimal blocking policy has a service level of .99925. This is only a .009% difference.

When the demand rate changes, lines controlled by kanban fill or empty their buffers, and the change in demand rate propagates upstream as blockages or starvations. The other policies change the material release rate to the system instantly as demand changes. Of course, no manager of a kanban line would allow extensive starvations or excessive inventories to occur as a consequence of a planned demand change. Instead she would change the numbers of kanban in circulation just before the demand rate is scheduled to change, so that the system response is improved. This would require much thought and adjustment to every kanban cell in the system.

In contrast, a line controlled by the hybrid policy has the consequences of a demand rate change localized to the finished goods buffer, as demonstrated by the trajectories in figure 3-4. This can then be counteracted more easily by changing the single WIP limit for the system. If the demand change is severe, the intermediate buffer sizes may need to be adjusted, but the consequences of maladjustments are small. For example, if all buffer sizes in a hybrid line are allowed to be infinite, the resulting control policy is CONWIP. This policy consistently comes in a close second best in our study.

3.2 A longer line

3.2.1 Purpose

We want to study a case more typical of Western manufacturing systems than the one in the previous section. The Toyota production system is characterized by low variability and short production lines, and we used a lost sales model to represent the assembly line. To lift these restrictions, we now investigate a longer line with more variability, where unlimited backlogging is permitted.

3.2.2 Model

We consider a 10-stage tandem line, where we allow unlimited backlogging if demand cannot be served from stock. We require that 75% of demand is to be served from stock as a service level target. Demands can occur only at integer multiples of the time unit. The demand process is assumed to be geometrically distributed with probability .5. That is, there is a probability of .5 that a demand event occurs at each whole time unit. We use a lognormal operation time distribution with mean 1.0 and CV .02. For all machines, we set the mean time to fail to 100 time units and the mean time to repair to 10 time units, using exponential distributions in both cases.

This system has nine internal buffers and the finished goods buffer. To simplify the optimization problem, we constrain the internal buffers, buffers 1 through 9, to have equal sizes, and let the finished goods buffer size be different. This leaves an optimization problem in two variables, which can be done using simulation. (A promising approach to optimizing the buffer allocations in similar systems is described in Schor (1995).)

3.2.3 Results

As in the four-machine line in section 3.1, there is no incentive to hold inventory in the intermediate buffers, since we penalize equally for WIP in all locations. It is therefore reasonable to look for solutions with small internal buffers and a large finished goods buffer, and solutions where the basestock levels of the internal buffers are zero.

Searching over buffer sizes 2, 3, and 4 for the internal buffers and 5 through 30 for the finished goods buffer, we find that the best kanban and minimal blocking configurations use internal buffers of size 2, combined with a finished goods buffer of size 21 for kanban and 14 for minimal blocking. Similarly, we find a inventory limit of 20 for CONWIP control and a finished goods basestock level of 18 for basestock control.

Policy	Service	WIP	Avg.backlog
Kanban	0.757 ± 0.008	26.4 ± 0.2	4.9 ± 0.5
Minimal block.	0.754 ± 0.006	23.84 ± 0.09	3.7 ± 0.2
Basestock	0.755 ± 0.003	19.58 ± 0.06	2.15 ± 0.06
Basestock/kanban	0.751 ± 0.003	19.53 ± 0.05	2.19 ± 0.06
Basestock/min.block.	0.755 ± 0.003	19.52 ± 0.05	2.14 ± 0.06
CONWIP	0.758 ± 0.004	19.44 ± 0.01	2.6 ± 0.1
CONWIP/min.block.	0.757 ± 0.004	19.36 ± 0.01	2.6 ± 0.1
CONWIP/kanban	0.752 ± 0.004	19.32 ± 0.01	2.8 ± 0.1

Table 3.3: Performance measures, 10-machine line

The best hybrid policies combine the CONWIP limit of 20 with internal buffers of size 9 for a CONWIP/kanban hybrid, and internal buffers of size 6 for a CONWIP/minimal blocking hybrid. Also, we combine the finished goods basestock level of 18 with internal buffers of size 15 for a basestock/kanban hybrid, and internal buffers of size 10 for a basestock/minimal blocking hybrid.

The results are shown in table 3.3, where the WIP includes all inventory that has been loaded on the first machine, but not yet delivered to satisfy demand. The average backlog is the time average amount of demand waiting. An alternate measure would be the average backlog conditioned on there being waiting demand. This will give much larger numbers, since the backlog is zero whenever the finished goods buffer contains material, which is a significant fraction of the time.

At the same levels of throughput and service, the CONWIP/kanban hybrid policy has 27% lower inventories and a very significant 43% lower average backlog compared to the best kanban control.

3.2.4 Discussion

There is a major difference between those controls that only have a local information flow — kanban and minimal blocking — and the rest that have a global information flow. On the other hand, there does not seem to be much to gain in this system by introducing the local information flows to the CONWIP or basestock policies. There are two possible explanations for this: Firstly, this line has a lower utilization than the one in section 3.1. Using the kanban control in saturating demand, this line has a maximal production rate of .568, which means that the demand rate of .5 gives about 88% utilization. We saw from figure 3-1 that utilization is an important factor in choosing between these policies. Since the system has relatively high variability, higher utilization rates require much higher inventories to achieve the service level requirement, but the differences between basestock and CONWIP on one hand and the different hybrid policies on the other could become larger. This should be explored further. Secondly, we have made all internal buffer sizes equal. In section 3.1, the best configurations had a larger buffers towards the end of the line. This holds some intuitive merit, since small buffers in the upstream portion of the system tend to block release of parts into the system, while the buffers at the end contribute more to the throughput by absorbing variability. It is possible that this line also would benefit from individually adjusted buffer sizes. We have not yet been able to optimize over all buffer sizes simultaneously, and this should also be explored further.

The lowest average backlog occurs with the basestock policy, which we think is due to the faster recoveries from failures in this policy. The average backlog increases with more stringent limitations on inventory build-up in front of a failed machine, where kanban has the strictest limitation of just 2 units. Other performance measures can also be calculated. It is straightforward to calculate average waiting times from the average backlog results by using Little's law

$$L = \lambda W \tag{3.1}$$

where L is the average queue length (backlog), λ the arrival (demand) rate, and W the average waiting time (Little 1961). Additional service quality statistics should be explored, such as the average waiting times given that the demand has to wait.

Since the effect of going from a CONWIP limit of 18 to 19, say, is large compared to the impact of going from one hybrid policy to another, we consider these results inconclusive in ranking the different hybrid control policies. Although the differences are likely to be small compared to the jump from kanban to CONWIP, this should also be explored further.

3.3 A semiconductor factory

3.3.1 Purpose

Semiconductor manufacturing differs from many other manufacturing environments by the complexity and variability of its processes. A semiconductor process can be several hundred operations long, and the machines are relatively unreliable due to the leading edge technology involved. This makes semiconductor factories harder to control than other plants of comparable size.

In the following, we report simulation results from datasets distributed by Sematech (Feigin, Fowler, Robinson, and Leachman 1994; Sematech 1995), representing actual semiconductor factories. These results should be seen as preliminary investigations into extending the control policies of the previous two sections to a very complex environment.

3.3.2 Methods

Control policies

The semiconductor factories we study make several kinds of products, and each product visits some machines several times. This requires us to specify two parts of the operating policy: The *release* policy that determines when new jobs are released to the factory, and the *dispatch* policy that determines the relative priorities of the jobs already in the factory.

A control policy called *periodic release* is often used in semiconductor factories. This releases new lots to the factory at equal time intervals, without any feedback from the current factory state. We combine this release policy with a dispatch policy that gives preference to lots with fewest remaining operations for a realistic reference control policy.

A simple way to regulate production based on the inventory state is *kanban* control, as defined in chapter 2. We introduce buffers of size 5 between all operations in each process. We then create a deterministic demand process to pull lots out of finished goods inventory with constant time intervals corresponding to the target production rate. If no lots are present as finished goods inventory, we backlog the demand until it can be satisfied. Lots are released whenever the first machine used in a process is idle and its output buffer for that product is not full. We combine this with a first come, first serve dispatch policy to determine which lot gets loaded on a machine that has several choices.

As mentioned in section 2.4, the *hedging point* control is related to the basestock control, but the production is regulated to track a constant demand rate instead of a stochastic demand process. We use this approach rather than the basestock control, since a semiconductor factory is operated on weekly or monthly production goals rather than in response to individual demand events. This policy also implies both a release and a dispatch policy: Lots are released to maintain a constant *surplus* ahead of the demand rate. This surplus value is called the hedging point. The same decision is made at every machine. If several lots are eligible for loading, the machine loads the oldest lot from the process that is farthest below its hedging point at this operation.

For this study, we compute a hedging point for the first machine in each process as $Z = (\text{demand rate}) * (\text{raw processing time})$, and set all other hedging points in the process equal to this value. This choice of parameters does not allow for queueing time, so to the extent it can be realized, the lots do not spend any time in buffers, but go directly from one station to be worked on at the next one. It is possible to adjust the hedging points using the throughput times generated by the simulation instead of the raw processing time for better backlog performance. A method for setting the hedging points was developed by Rubio and Wein (1994). The periodic release control can be seen as a special case of the hedging point control with all hedging points set to zero.

Similarly, we use a *two-boundary* control policy that combines the hedging point control with finite buffers between the machines. This causes an upstream station to avoid a particular product type if the downstream buffer for that type is full, while applying hedging point control to all other product types. We again used buffers of size 5 between all production stages.

The CAFE scheduling and simulation testbed

The simulator used in sections 3.1 and 3.2 is only suitable for simple tandem lines with a single product type, and not for a complicated production environment like a semiconductor factory. We have built a scheduling and simulation testbed integrated with the MIT CAFE computer aided fabrication environment (Bonvik 1994; McIlrath, Troxel, Boning, Heytens, Penfield, and Jayavant 1990). This program can act as a factory floor dispatcher, where it recommends a daily schedule based on the factory state stored in the database. It is used for this purpose in the MIT Microsystems Technologies Laboratories semiconductor manufacturing facilities. It can also run in a simulation mode, where it projects long-term consequences of following a certain scheduling policy under various assumptions on lot releases and machine reliabilities.

In its simulation mode, the testbed can read the Sematech datasets instead of the CAFE

Dataset no.	1	3
Number of products	2	11
Equipment groups	83	73
Number of machines	265	354
Raw process time	313 - 356 hrs	173 - 359 hrs
Process steps	210 - 245	270 - 473

Table 3.4: Summary of the Sematech sample fabs

database. This allows us to test out our scheduling approaches in other manufacturing systems than our own. The testbed does not currently support all features found in the Sematech datasets. For instance, we do not simulate operator availability, scrap, or rework. Setups are simulated, but the scheduling policies tested do not make any attempt to optimize the setup sequence.

This leads to different production rates from our testbed than from the Delphi simulator also distributed by Sematech. On the other hand, we allow the scheduling modules to control the lot release rates to the fab, not just manipulate the dispatch decisions as appears to be the case with Delphi. Previous work (Wein 1988) has shown that the lot releases are a more important factor in fab performance than lot dispatch decisions on the shop floor.

3.3.3 Results

We have concentrated on Sematech datasets 1 and 3. These fabs are summarized in Table 3.4. Our testbed gives different results from the actual fab, and we estimated the maximal throughput of the testbed model as follows: We introduced buffers of size 10 between all operations, and ran the fabs by a kanban control, using a demand rate that was surely too high for the plant. The resulting production rate was used as the maximal throughput estimate. We reduced the release rates to 95% of these values for all products to create a heavily utilized, but not unstable environment.

For each sample fab and every control policy, we ran 30 replications, each for 52 weeks of 7-day, 24-hour operation, using an initial warm-up period of 8 weeks. The performance measures were averaged over all replications, and 95% confidence intervals estimated as $\overline{X} \pm 1.96S/\sqrt{N}$, where $\overline{X}$ is the sample mean, S the sample standard deviation, and N the sample size.

Fab 1

In our simulations, fab 1 had a production target of 38.14 lots per week, consisting of 27.49 lots of product 1 and 10.65 lots of product 2. The raw processing times are 1.86 and 2.11 weeks, respectively. This gives a minimum WIP of 73.6 lots by Little's law (equation 3.1). The variability in the fab causes queuing delays, so the actual throughput times and WIP levels is higher.

Some overall performance measures of interest are the work in progress and lot completions per week, summed over all products. We are also interested in smoothness of output,

Control policy	Avg. WIP (lots)	Avg. lots/week	Std.dev. lots/week
Periodic rel.	101.6 ± 0.6	$37.93 \pm .05$	$2.16 \pm .08$
Kanban	337.5 ± 1.8	$39.91 \pm .04$	$1.75 \pm .05$
Hedging pt.	107.7 ± 0.4	$38.92 \pm .02$	$2.43 \pm .07$
Two-bound.	94.1 ± 0.9	$39.28 \pm .04$	$1.85 \pm .05$

Table 3.5: Overall performance measures for Fab 1

Control policy	Product	Throughput time
Periodic rel.	1	$2.53 \pm .01$
	2	$3.02 \pm .02$
Kanban	1	$6.97 \pm .06$
	2	$10.58 \pm .04$
Hedging pt.	1	$2.55 \pm .01$
	2	$3.54 \pm .01$
Two-bound.	1	$2.21 \pm .01$
	2	$3.09 \pm .07$

Table 3.6: Throughput times in weeks for Fab 1

which can be measured by the standard deviation of the number of lots completed per week. These performance measures are shown in Table 3.5. We note that the kanban control has the highest WIP by far, while the two-boundary control has the least WIP.

We also note that the average output per week for the periodic release policy falls somewhat short of the target, while the other policies exceed the target over the measured period. This seems to indicate that the periodic release policy is still accumulating inventory, since the output must equal the input in the long run. The hedging point and two-boundary control work at capacity for some length of time before slowing down to the demand rate, so this drives its production rate up. Kanban backlogs demand during the initial phase, and tries to catch up when finished goods start appearing. This leads to a similar behavior as the hedging point control in this respect, but it also attempts to keep some finished goods in inventory, so its production rate is even higher.

Another performance measure of great interest is the throughput time for each product type. We define this as the time from when the lot is released to the fab until it leaves the last machine in its process. These results are shown in Table 3.6. The results closely conform to the results for WIP, as one would expect. The two-boundary policy has the shortest throughput time for product 1 and the second shortest for product 2.

Control policy	Avg. WIP (lots)	Avg. lots/week	Std.dev. lots/week
Periodic rel.	46.5 ± .01	27.01 ± .01	1.43 ± .04
Kanban	818.5 ± 1.2	28.67 ± .03	7.90 ± .32
Hedging pt.	73.6 ± .02	27.07 ± .01	1.50 ± .06
Two-bound.	73.6 ± .02	27.07 ± .01	1.50 ± .05

Table 3.7: Overall performance measures for Fab 3

Control policy	Avg. WIP (lots)	Avg. lots/week	Std.dev. lots/week
Kanban 2	499.6 ± 1.2	27.63 ± .02	3.17 ± .08
Kanban 1	401.3 ± 1.1	27.25 ± .01	1.81 ± .05
Two-boundary 2	62.5 ± 0.6	27.07 ± .01	1.48 ± .05
Two-boundary 1	57.3 ± .01	27.08 ± .01	1.47 ± .07

Table 3.8: Fab 3, alternate buffer sizes

Fab 3

Fab 3 had a production target of 27.05 lots per week, divided among 11 product types. Table 3.7 shows the overall performance measures for this fab.

As for fab 1, the periodic release policy falls slightly short of the production target, and the kanban policy builds excessive inventories. It is clear that the buffer sizes of 5 are too large for this system, since the two-boundary policy gives the same results as the hedging point policy. This indicates that the buffer sizes never become a constraint in the simulation. To test this hypothesis, we ran more simulations with the kanban and two-boundary policies, this time with buffer limits of 2 and 1 lots. We reduced the number of replications to 10 for these additional explorations. The results are shown in Table 3.8. Even with buffer sizes of 1, kanban has much higher inventories than the other policies.

Kanban had the least standard deviation of lot completions per week in Fab 1, but has the highest output variability in this fab. Also, the two-boundary policy had a lower output variability than periodic release in Fab 1, but higher in this one. This makes it hard to draw conclusions based on this performance measure.

Workload sensitivity

We tested the sensitivity to fab utilization rates by rerunning the experiments with production targets at 90% and 98% of the sustainable production rate instead of 95%, again using 10 replications per case. The results are in Tables 3.9 and 3.10. The production targets are 36.14 and 39.35 lots per week for Fab 1 at 90% and 98% utilization, respectively. For Fab 3, the targets are 25.62 and 27.91 lots per week. We note a similar pattern as in the previous

Control policy	Avg. WIP (lots)	Avg. lots/week	Std.dev. lots/week
Periodic rel. 90%	78 ± 4.5	36.10 ± .05	1.6 ± .5
Kanban 90%	383 ± 2.4	37.62 ± .05	2.2 ± .1
Hedging pt. 90%	90 ± 0.5	36.79 ± .03	2.6 ± .1
Two-bound. 90%	79 ± 0.3	36.91 ± .01	2.0 ± .1
Periodic rel. 98%	112 ± 0.7	39.09 ± .05	2.1 ± .2
Kanban 98%	335 ± 3.6	40.42 ± .04	1.6 ± .2
Hedging pt. 98%	126 ± 0.7	40.13 ± .04	2.4 ± .2
Two-bound. 98%	116 ± 2.2	40.51 ± .05	1.7 ± .2

Table 3.9: Fab 1, alternate demand rates

Control policy	Avg. WIP (lots)	Avg. lots/week	Std.dev. lots/week
Periodic rel. 90%	44 ± .01	25.58 ± .01	1.5 ± .1
Kanban 90%	860 ± 3.1	27.15 ± .06	8.5 ± .4
Hedging pt. 90%	71 ± .05	25.65 ± .01	1.4 ± .1
Two-bound. 90%	71 ± .04	25.64 ± .01	1.4 ± .1
Periodic rel. 98%	48 ± .01	27.92 ± .01	1.5 ± .1
Kanban 98%	794 ± 2.2	29.67 ± .06	7.6 ± .7
Hedging pt. 98%	75 ± .03	28.05 ± .02	1.9 ± .1
Two-bound. 98%	75 ± .02	28.06 ± .02	2.0 ± .1

Table 3.10: Fab 3, alternate demand rates

experiments, where the periodic release policy falls slightly short of the production target (except in Fab 3 at 98%) and the kanban policy has very high inventories.

The WIP results from the periodic release policy in Fab 1 at 90% have a very high sample variance, reflected in the wide confidence intervals. This is because the individual simulation results were close to either 73 or 88 lots of WIP. No other case exhibited this bimodal behavior.

3.3.4 Discussion

Kanban is not a good choice of control policy for semiconductor manufacturing. Although kanban control imposes inventory limits at all stages in the production process, it also attempts to build inventory right up to these limits. In a production system characterized by many long processes, this leads to a large number of small inventories, adding up to excessive work-in-progress. It is worth noting that the inventories increase as the demand rate decreases in a system controlled by kanban, while they decrease in the other control policies we tested.

Another problem is that kanban control amplifies variability upstream in the system, as shown in figure 3-3 for the four-machine line. In very long production processes, like in semiconductor manufacturing, production stages far upstream alternate between periods of intense production pressure and periods of blockage. In contrast, the other policies regulate the lot release process, creating a more even workload at the upstream end of the process.

We have not tried to optimize the parameter choices of the different policies. It is probably the case that somewhat better results can be obtained by other distributions of buffer sizes in the kanban policy. Finding the optimal choice of all buffer sizes would be an enormous undertaking and would require very expensive computational resources, due to the size of the manufacturing systems involved. The hedging point and two-boundary controls are mainly controlled by the single hedging point of the first machine, with some minor adjustments possible by changing the other parameters of the system. This is much easier to optimize and adjust in practice. Still, efficient tools for setting the parameters of these policies would be very useful.

The hedging point and two-boundary policies also allow for the just-in-time practice of forced improvements, where management reduces the inventory limits of the system gradually to uncover and solve production problems and to benefit from the learning that has already occurred in the organization. This can be done by adjusting the overall inventory limit, which is set by the hedging point of the first stage in each process, rather than adjusting every single buffer size as in a kanban system.

The hedging point and two-boundary policies are more complex to implement than the periodic lot release policy. It is necessary to keep track of the production surplus for each product at each station. This quantity is the cumulative production for the station minus cumulative demand. In addition, the two-boundary control requires information about the queue length at the next downstream station for each process.

3.4 Conclusions

We have studied the performance of kanban, minimal blocking, basestock, CONWIP, and hybrid control policies in simulations of a variety of production systems.

In all cases, the hybrid control policy demonstrated superior performance in achieving a high service level with minimal inventories. The advantage over kanban increases with the size of the system and the amount of process variability, such as machine failures, that is present in the system. The advantage over CONWIP increases with system utilization, but is only significant in system utilization rates over about .9.

The hybrid policy we studied can be implemented as a straightforward modification to a kanban policy, simply by routing kanban from the finished goods buffer to the first production stage instead of the last. It can also be implemented as a modification of a CONWIP system by introducing kanban cells to regulate inventories upstream of a bottleneck stage.

Chapter 4

Building blocks for decomposition

The simulation results in the previous chapter were acquired by using vast amounts of computer time, as described in appendix A. Even when using a dozen UNIX workstations in parallel, these studies took several days to run. Such computation-intensive methods are fine for limited research purposes, but are prohibitive for practical manufacturing systems design.

The goal of this and the following chapters is to develop methods that allow at least approximate analysis of production systems operated by these control policies in a few seconds on a personal computer. We hope this will bring these techniques closer to industrial acceptance and use.

To construct fast methods for analysis, we focus on the principle of *decomposition*. In a decomposition method, an intractable problem is broken down into tractable building blocks, which are then assembled to a representation of the original system.

We use a continuous material or fluid representation of the system, where the flow of material is approximated by a continuous flow that can be interrupted by machine failures. This is a useful representation of an asynchronous system with unreliable machines, but where the operation times are deterministic or nearly deterministic and much shorter than the mean up- and downtimes of the machines (Alvarez Vargas, Dallery, and David 1991b). By "asynchronous" we mean that the individual machines in the system are not bound to a common operation cycle, so they can start an operation whenever the machine is not blocked or starved (Dallery and Gershwin 1992).

First, we give a brief example of the decomposition principle to motivate the derivation of building blocks. The rest of this chapter derives the solutions of the necessary building blocks, and in the following chapters we assemble these to a system representation and derive an algorithm for solving the resulting equations.

4.1 Decomposition principles

The principle of decomposition is to break an analytically intractable system into a set of tractable components and equations relating the components that together approximate the whole system. Within this principle, there is a lot of room for engineering judgement in what features are essential and what details to leave out.

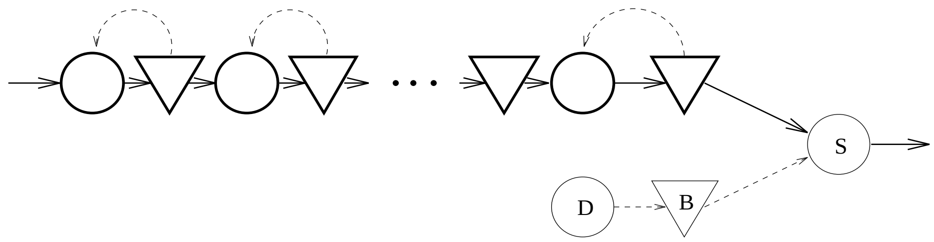

Figure 4-1: Kanban control with backlogging

We use a decomposition approach very similar to that of Gershwin (1987a), Dallery, David, and Xie (1988), and several other of the papers cited in section 1.1.2. In this approach, the overall behavior of a production system is representing by focusing on the behavior of individual buffer inventories. Even if there is a complicated production system upstream of a particular buffer, an observer in the buffer will see a flow pattern into the buffer that is quite similar to that caused by a single unreliable machine. The flow out of the buffer behaves the same way. Therefore, we can approximate the behavior of the system by decomposing it into a collection of two-machine lines, where each machine represents the pattern of flow into or out of a particular buffer.

In contrast to most of the previous papers, we want to represent systems that do not just consist of tandem machines and buffers. A kanban system is a simple example. The line itself behaves like a tandem line of machines and buffers, but the assumption that the last machine is never blocked does not hold. Instead, the last machine is synchronized with an external demand process and is blocked whenever it has caught up with the demand. This is illustrated in figure 4-1. In the figure, machine D is the demand process. It delivers demand information to the synchronization machine S. This machine joins equal parts of demand and products from the line. Any demand that cannot be instantly met waits in buffer B, which is the system backlog. The production line itself is the upper sequence of machines and buffers in the figure, and it is synchronized with the demand by kanban circulating between each buffer and its immediate upstream machine.

We note that with the assumptions we made when defining the kanban control in section 2.2, the information flow from a buffer to the upstream machine occurs when pieces are taken out of the buffer, and an empty space in the buffer corresponds to a kanban waiting at the machine. We can therefore use a tandem two-machine line with a finite buffer to represent the behavior of each kanban loop.

The final stage is different. This stage consists of the synchronization machine S, the two buffers feeding it, and the machine immediately upstream of each buffer. We represent the synchronization of products and demand as a fast and reliable operation joining items from two buffers. This is an assembly operation, which has been studied by Liu and Buzacott (1990), Gershwin (1991) and Di Mascolo, David, and Dallery (1991). For now, we represent this operation as the three-machine two-buffer assembly system shown in figure 4-2, where we have illustrated how the kanban controlled line can be decomposed.

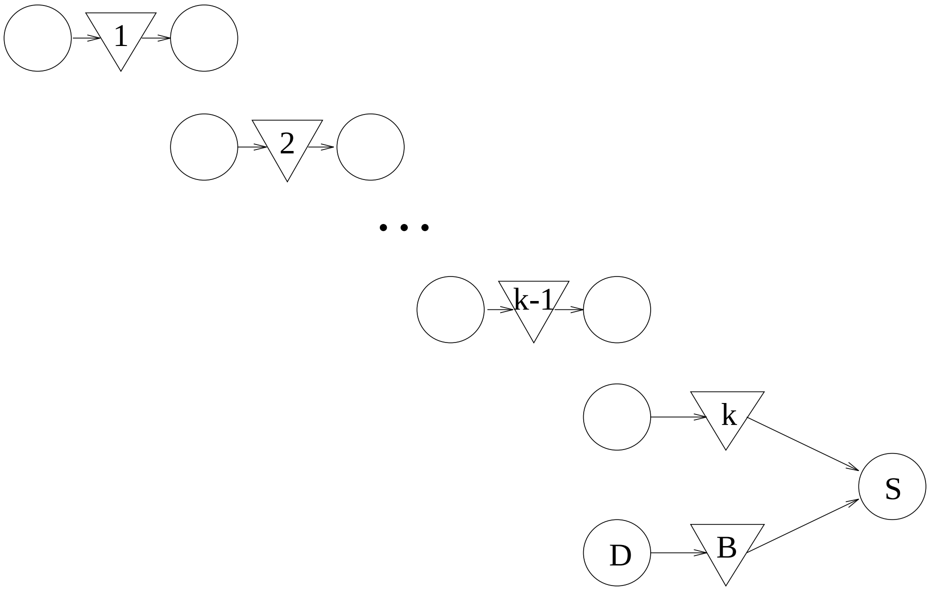

Figure 4-2: Decomposition of kanban controlled line

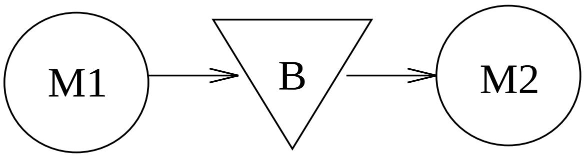

Figure 4-3: A two-machine line

4.2 Tandem two-machine line

4.2.1 Finite buffer case

This is the only previously existing building block, and we refer the reader to Gershwin (1994), section 3.10, for the details. We refer to specific equations in that book by prefixing the equation number with a letter G, for example (G3.200). In the following, we provide an extremely brief outline of the solution procedure for comparison with the other building blocks developed in sections 4.2.2 and 4.3.

We consider a two-machine tandem line, where the machines are separated by a finite buffer, as shown in figure 4-3. We assume that the first machine in the line has an unlimited supply of raw material, and that the second machine has unlimited demand for finished products, so any interruptions of the flow through the buffer are due to the machines themselves. We also assume that the times to fail and repair are exponentially distributed, and that failures are *operation dependent*, meaning that the failure rate is proportionally decreased whenever a machine is working at a rate below its maximal processing rate.

First we need to introduce some notation: The parameters that describe the line are

r_i, the repair rate of machine i, for $i = 1, 2$.

x	α_1	α_2	u_1	u_2
N	1	1	$\min\{\mu_1, \mu_2\}$	μ_2
$0 < x < N$	1	1	μ_1	μ_2
0	1	1	μ_1	$\min\{\mu_1, \mu_2\}$
N	1	0	0	0
$0 < x < N$	1	0	μ_1	0
0	1	0	μ_1	0
N	0	1	0	μ_2
$0 < x < N$	0	1	0	μ_2
0	0	1	0	0
N	0	0	0	0
$0 < x < N$	0	0	0	0
0	0	0	0	0

Table 4.1: Input and output rates of the tandem two-machine line

p_i, the failure rate of machine i, for $i = 1, 2$.
μ_i, the maximal processing rate of machine i, for $i = 1, 2$.
N, the capacity of the buffer.

The state of the system at time t is described by

$\alpha_i(t)$, the repair state of machine i, $\alpha_i \in \{0, 1\}$ for $i = 1, 2$.
$x(t)$, the level of material in the buffer, $x \in [0, N]$.

We also define $u_i(t)$ to be the instantaneous production rate of machine i, $u_i \in [0, \mu_i]$ for $i = 1, 2$. The state variables α_1, α_2, and x are sufficient to describe the system. This, together with the system control policy, determines the values of u_1 and u_2 as shown in table 4.1, which is constructed as follows: We assume that each machine is working at its maximal rate if it is not failed and the buffer is not full or empty. If the buffer is full, machine 1 works at rate $u_1 = \min\{\alpha_1\mu_1, \alpha_2\mu_2\}$. Similarly, if the buffer is empty, machine 2 works at rate $u_2 = \min\{\alpha_2\mu_2, \alpha_1\mu_1\}$. This way, the buffer is the only synchronization mechanism between these machines. When machine 1 is forced to slow down due to a full buffer, we say that it is *blocked*, and when machine 2 is forced to slow down due to an empty buffer, we say that it is *starved*.

A complicating factor in this model is that *partial* starvations and blockages can occur. A partial blockage occurs when the buffer downstream of a machine m is full, but there is a flow out of the buffer at some rate u that is strictly less than the maximal production rate μ_m of machine m. As a consequence, m slows down to work at rate u. Due to the operation dependent failures, this requires the failure rates to be adjusted accordingly. We assume that the instantaneous failure rate is a linear function of the working speed, or $p'_m = p_m u/\mu_m$. The definition of partial starvation is similar.

The state probability distribution for this system consists of four densities across all levels of the buffer (one for each combination of machine repair states), four impulses at the buffer full boundary, and four impulses at the buffer empty boundary. We write the densities as $f(x, \alpha_1, \alpha_2)$ and the impulses as $\mathbf{p}(x, \alpha_1, \alpha_2)$.

When the buffer is not full or empty, the buffer level changes at rate $\alpha_1\mu_1 - \alpha_2\mu_2$. We can then write four partial differential equations that describe the development in time of the buffer level for all combinations of machine repair states (Gershwin 1994, equations (G3.197) - (G3.200)). Similarly, we can write boundary conditions that describe the behavior of the system when the buffer is full or empty. Setting the time derivatives to zero in order to get the steady state distribution, we arrive at a set of coupled ordinary differential equations and boundary conditions. We also have a normalization equation (G3.222) that states that the probabilities integrate to unity. From some of the boundary conditions, we have

$$\mathbf{p}(0,0,0) = \mathbf{p}(0,1,0) = \mathbf{p}(N,0,0) = \mathbf{p}(N,0,1) = 0 \tag{G3.226}$$

and, if $\mu_1 < \mu_2$

$$\mathbf{p}(N,1,1) = 0 \tag{G3.278}$$

while, if $\mu_1 > \mu_2$

$$\mathbf{p}(0,1,1) = 0 \tag{G3.268}$$

From the form of the partial differential equations, we assume a solution of the form

$$f(x, \alpha_1, \alpha_2) = Ce^{\lambda x} Y_1^{\alpha_1} Y_2^{\alpha_2} \tag{G3.255}$$

where the parameters C, λ, Y_1, and Y_2 must be found. Plugging this solution back into the differential equations, we arrive at three equations in three unknowns called the *parametric equations*

$$\sum_{i=1}^{2} (p_i Y_i - r_i) = 0 \tag{G3.256}$$

$$-\mu_1 \lambda = (p_1 Y_1 - r_1) \frac{1 + Y_1}{Y_1} \tag{G3.257}$$

$$\mu_2 \lambda = (p_2 Y_2 - r_2) \frac{1 + Y_2}{Y_2} \tag{G3.258}$$

By solving these for Y_1 and removing a trivial solution where $\lambda = 0$, we arrive at a single quadratic equation in Y_1

$$-(\mu_2 - \mu_1) p_1 Y_1^2 + [(\mu_2 - \mu_1)(r_1 + r_2) - (\mu_2 p_1 + \mu_1 p_2)] Y_1 + \mu_2 (r_1 + r_2) = 0 \tag{G3.259}$$

Solving equation (G3.259) for Y_1, we can also determine λ and Y_2 from the remaining two parametric equations. There are now two cases, depending on the relationship between μ_1 and μ_2. If the machines are equally fast ($\mu_1 = \mu_2$), equation (G3.259) reduces to a linear equation. Otherwise, there are two solutions, and the probability densities are expressed as a sum of terms from both solutions

$$f(x, \alpha_1, \alpha_2) = \sum_{j=1}^{2} C_j e^{\lambda_j x} Y_{1j}^{\alpha_1} Y_{2j}^{\alpha_2} \tag{G3.267}$$

If $\mu_1 > \mu_2$, $\mathbf{p}(0,1,1) = 0$, which requires that $f(0,1,0) = 0$ (G3.212), or

$$C_1 Y_{11} + C_2 Y_{12} = 0 \tag{G3.270}$$

Using this equation and the normalization equation, which has the form

$$A_1 C_1 + A_2 C_2 = 1 \tag{G3.274}$$

we can solve for the normalization constants C_1 and C_2 and thus complete the solution.

If $\mu_1 < \mu_2$, we use a very important relationship called *line reversibility*: The behavior of *material* in the buffer of the line consisting of machine 1 followed by machine 2 is the same as the behavior of *empty space* in the buffer of a line consisting of machine 2 followed by machine 1. This is true for lines with finite buffers (Buzacott and Shantikumar 1993, especially problem 5.18). We can therefore reverse the line, apply the previous case, and correct for the material/space relationship when calculating the performance measures.

4.2.2 Infinite buffer case

This case is an extension of the previous one that is needed in our decompositions. With an infinite buffer, stability becomes an issue. The *isolated production rate* of a machine is the average production rate it obtains if it is never blocked or starved. This is $\rho = \mu r/(r+p)$ (Gershwin 1994). If the isolated production rate of machine 1, which is never blocked or starved, is higher than or equal to that of machine 2, the buffer level goes to infinity. This stability condition can be written

$$\frac{\mu_1 r_1}{r_1 + p_1} < \frac{\mu_2 r_2}{r_2 + p_2}$$

and in the following, we assume that this is satisfied.

We get the same differential equations as in the finite buffer line, and the same boundary conditions for $x = 0$. However, the boundary conditions at $x = N$ no longer exist. The parametric equations and the resulting quadratic equation in Y_1 (equations G3.256 - G3.259) are the same as in the finite buffer line. If $\mu_1 = \mu_2$, the quadratic equation again reduces to a linear equation. If $\mu_1 > \mu_2$, we can find the impulses $\mathbf{p}(0,1,1)$ and $\mathbf{p}(0,0,1)$ from equations (G3.268) and (G3.269). The internal densities (for $x > 0$) are found from equation (G3.267). As in the finite buffer case, finding the two normalization constants C_1 and C_2 depends on equation (G3.270), which states that $f(0,1,0) = 0$, and the normalization equation (G3.222). Note that $\mu_1 > \mu_2$ does not contradict stability, since the "faster" machine 1 can be down a significant fraction of the time, and thus have a lower isolated production rate than the "slower" machine 2.

We now need to consider the case $\mu_1 < \mu_2$. We solve for Y_1, Y_2, and λ as in the finite buffer line. Note that, in this case, $\mathbf{p}(0,1,1) > 0$, so $f(0,1,0) > 0$, and we lose one of the equations that was used to find the normalization constants. It turns out that for this case, one λ is always positive and the other negative.[1] But we know that the system is stable, and

[1] We have not been able to prove this property rigorously, but we have computed thousands of randomly generated cases, and it has held in every single one.

the densities $f(x, \alpha_1, \alpha_2)$ exist. But the normalization equation (G3.222) contains terms of the form

$$\int_0^\infty f(x, \alpha_1, \alpha_2)\, dx = \int_0^\infty \sum_{j=1}^{2} C_j e^{\lambda_j x} Y_{1j}^{\alpha_1} Y_{2j}^{\alpha_2}\, dx$$

These integrals do not converge if $\lambda > 0$, which contradicts the existence of a density. We can therefore discard the positive solution for λ, and the corresponding normalization factor must be 0. We then get a solution consisting of a single term, and we can use the single normalization equation to complete the solution.

4.2.3 Calculating performance measures

We calculate the same performance measures as Gershwin (1994). These are

P, the average production rate of the line
$\bar{x}$, the average level of material in the buffer
p_s, the fraction of time machine 2 is starved
p_b, the fraction of time machine 1 is blocked

We also need to calculate a few additional performance measures from the tandem two-machine solution. The decomposition that we present in the next chapter needs the fractions of time each machine is working at each of the possible rates μ_1 and μ_2. We define

p_{i1}, the fraction of time machine 1 is working at rate μ_1
p_{i2}, the fraction of time machine 1 is working at rate μ_2
p_{o1}, the fraction of time machine 2 is working at rate μ_1
p_{o2}, the fraction of time machine 2 is working at rate μ_2

(Mnemonic: the first subscript refers to the *input* or *output* rate to/from the buffer.) There are three cases. If $\mu_1 < \mu_2$, from table 4.1 and the boundary conditions, we have

$$\begin{aligned}
p_{i1} &= \mathbf{p}(0,1,1) + \int_0^N f(x,1,1)\, dx + \int_0^N f(x,1,0)\, dx \\
p_{i2} &= 0 \\
p_{o1} &= \mathbf{p}(0,1,1) \\
p_{o2} &= \int_0^N f(x,1,1)\, dx + \int_0^N f(x,0,1)\, dx
\end{aligned}$$

If $\mu_1 = \mu_2$, we have

$$\begin{aligned}
p_{i1} &= \mathbf{p}(N,1,1) + \mathbf{p}(0,1,1) + \int_0^N f(x,1,1)\, dx + \int_0^N f(x,1,0)\, dx \\
p_{i2} &= 0 \\
p_{o1} &= 0 \\
p_{o2} &= \mathbf{p}(N,1,1) + \mathbf{p}(0,1,1) + \int_0^N f(x,1,1)\, dx + \int_0^N f(x,0,1)\, dx
\end{aligned}$$

This is a definition, since $\mu_1 = \mu_2$, and we cannot distinguish whether the input rate to the buffer, say, is μ_1 or μ_2. And, finally, if $\mu_1 > \mu_2$, we have

$$\begin{aligned}
p_{i1} &= \int_0^N f(x,1,1)\,dx + \int_0^N f(x,1,0)\,dx \\
p_{i2} &= \mathbf{p}(N,1,1) \\
p_{o1} &= 0 \\
p_{o2} &= \mathbf{p}(N,1,1) + \int_0^N f(x,1,1)\,dx + \int_0^N f(x,0,1)\,dx
\end{aligned}$$

In the infinite buffer case, the term $\mathbf{p}(N,1,1)$ disappears, since the boundary condition it is derived from does not exist. The integrals then range from zero to infinity, over all attainable buffer levels x.

It is also possible to calculate some measures of the production variability. Note that the input and output rates of the buffer can take on either of three discrete values: 0, μ_1, and μ_2. We now view these rates as random variables, say U_1 for the input rate and U_2 for the output rate. These are the same as the instantaneous production rate of machines 1 and 2 in the two-machine line. The random variables U_1 and U_2 have discrete distributions, consisting of impulses at 0, μ_1, and μ_2, or

$$P\{U_1 = u\} = \begin{cases} p_{i1} & \text{for } u = \mu_1 \\ p_{i2} & \text{for } u = \mu_2 \\ 1 - p_{i1} - p_{i2} & \text{for } u = 0 \\ 0 & \text{otherwise} \end{cases}$$

The distribution for U_2 is similar. Recall that P is the average production rate of the line. From conservation of flow, it must also be the average value of the production rates of the individual machines, or

$$E[U_1] = E[U_2] = P$$

From the definition of variance, we can then calculate

$$\mathrm{Var}[U_1] = (1 - p_{i1} - p_{i2})P^2 + p_{i1}(\mu_1 - P)^2 + p_{i2}(\mu_2 - P)^2 \tag{4.1}$$
$$\mathrm{Var}[U_2] = (1 - p_{o1} - p_{o2})P^2 + p_{o1}(\mu_1 - P)^2 + p_{o2}(\mu_2 - P)^2 \tag{4.2}$$

It is not clear, however, how to use these measures in practice. A more intuitive measure of variability is the variance of production over a certain period of time. This was explored by Carrascosa (1995), but seems to be much harder to derive than this simple measure.

4.2.4 Validation

We have implemented these calculations in a C program. Since these results are crucial to the decompositions we build on top of them, we want to validate that the results are correct. We do this three ways:

- The code is written defensively, so the same quantities are calculated in several ways and the results compared. If they do not agree, the program stops with a diagnostic message.

- We have run test cases to ensure continuity of performance measures across the various cases. For instance, suppose we hold all parameters except μ_1 fixed. We then expect to see similar results for $\mu_1 = \mu_2 - \epsilon$, $\mu_1 = \mu_2$, and $\mu_1 = \mu_2 + \epsilon$ for very small values of ϵ. This holds whenever applicable, and provides a cross-check on the solutions for the different cases derived in the previous section.

- We validate against simulations, which we now describe in more detail.

The code implementing these calculations has six major cases. These are $\mu_1 < \mu_2$, $\mu_1 = \mu_2$, and $\mu_1 > \mu_2$ for finite and infinite buffer, following the outline in sections 4.2.1 and 4.2.2. We therefore compared the analytic results with a simulation of the continuous fluid model of an unreliable two-machine line. For each major case, we constructed a line exemplifying the case. We then did $n = 15$ simulation runs or *replications* of the line, each for 10^7 time units with an initial warm-up period of 1000 time units. For each replication, the random number generators were initialized from a different seed, so the batch of replications can be seen as an independent sample of size n. For each performance measure listed in section 4.2.3, we calculated the sample mean and variance across the 15 replications, and calculated a 95% confidence interval using a Student's t-distribution with $n-1 = 14$ degrees of freedom, or

$$x \in \bar{X} \pm t_{14}^{.025} \frac{S^2}{\sqrt{n}} \tag{4.3}$$

where x is some performance measure, $\bar{X}$ the sample mean of that performance measure, S^2 the sample variance, and n the sample size. This assumes that the distribution of each performance measure across the replications can be reasonably approximated by a normal distribution (Arnold 1990). Equation (4.3) becomes

$$x \in \bar{X} \pm 2.145S^2/3.873 = \bar{X} \pm 0.554S^2$$

The validation results are summarized in tables 4.2 through 4.5. We have divided case 1, where the machines have equal processing rates ($\mu_1 = \mu_2$) into two subcases 1a and 1b, because identical machines cause λ to become zero, and this case is handled separately in the code. For each case, we posed the hypothesis that the analytical model gives the correct value of the performance measure in question. This hypothesis is rejected at the .05 confidence level if the analytical value falls outside the 95% confidence interval of the corresponding simulated performance measure as determined by equation 4.3. If the analytical performance measure falls within the confidence interval, we cannot reject the hypothesis. This is summarized in the last column of each table, where the term "fail" is used in cases where the hypothesis was rejected, and the term "pass" when it was not rejected.

We note that the only validation failures that occured appear in table 4.7. Redoing the experiment, these performance measures pass, and occasional failures occur elsewhere. We conclude that these are spurious failures from our statistical procedure, since testing at the .05 confidence level means that on the average, correct hypotheses will be rejected in 5% of the cases.

r_1	p_1	μ_1	N	r_2	p_2	μ_2
.1	.01	1.0	10	.1	.01	1.0

Perf.meas.	Simulation	Analytic	Verdict
P	0.8586 $\pm$ 0.0003	0.8587	pass
$\bar{x}$	4.9954 $\pm$ 0.0083	5.0000	pass
p_b	0.0554 $\pm$ 0.0001	0.0554	pass
p_s	0.0555 $\pm$ 0.0002	0.0554	pass
p_{i1}	0.8586 $\pm$ 0.0003	0.8587	pass
p_{i2}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o1}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o2}	0.8586 $\pm$ 0.0003	0.8587	pass
$\mathbf{p}(0,0,1)$	0.0555 $\pm$ 0.0002	0.0554	pass
$\mathbf{p}(0,1,1)$	0.2777 $\pm$ 0.0008	0.2770	pass
$\mathbf{p}(N,1,0)$	0.0554 $\pm$ 0.0001	0.0554	pass
$\mathbf{p}(N,1,1)$	0.2770 $\pm$ 0.0006	0.2770	pass

Table 4.2: Case 1a: Finite buffer, identical machines

r_1	p_1	μ_1	N	r_2	p_2	μ_2
.1	.01	1.0	10	.1	.1	1.0

Perf.meas.	Simulation	Analytic	Verdict
P	0.4937 $\pm$ 0.0002	0.4936	pass
$\bar{x}$	9.3463 $\pm$ 0.0027	9.3461	pass
p_b	0.4570 $\pm$ 0.0003	0.4571	pass
p_s	0.0128 $\pm$ 0.0001	0.0129	pass
p_{i1}	0.4937 $\pm$ 0.0002	0.4936	pass
p_{i2}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o1}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o2}	0.4937 $\pm$ 0.0002	0.4936	pass
$\mathbf{p}(0,0,1)$	0.0128 $\pm$ 0.0001	0.0129	pass
$\mathbf{p}(0,1,1)$	0.0117 $\pm$ 0.0001	0.0117	pass
$\mathbf{p}(N,1,0)$	0.4570 $\pm$ 0.0003	0.4571	pass
$\mathbf{p}(N,1,1)$	0.4156 $\pm$ 0.0003	0.4155	pass

Table 4.3: Case 1b: Finite buffer, $\mu_1 = \mu_2$, not identical machines

r_1	p_1	μ_1	N	r_2	p_2	μ_2
.1	.01	1.1	10	.1	.01	1.0

Perf.meas.	Simulation	Analytic	Verdict
P	0.4943 ± 0.0002	0.4944	pass
$\bar{x}$	9.4943 ± 0.0024	9.4945	pass
p_b	0.5056 ± 0.0003	0.5056	pass
p_s	0.0113 ± 0.0001	0.0112	pass
p_{i1}	0.0561 ± 0.0002	0.0561	pass
p_{i2}	0.4326 ± 0.0003	0.4326	pass
p_{o1}	0.0000 ± 0.0000	0.0000	pass
p_{o2}	0.4943 ± 0.0002	0.4944	pass
$\mathbf{p}(0,0,1)$	0.0113 ± 0.0001	0.0112	pass
$\mathbf{p}(0,1,1)$	0.0000 ± 0.0000	0.0000	pass
$\mathbf{p}(N,1,0)$	0.4663 ± 0.0003	0.4663	pass
$\mathbf{p}(N,1,1)$	0.4326 ± 0.0003	0.4326	pass

Table 4.4: Case 2: Finite buffer, $\mu_1 > \mu_2$

r_1	p_1	μ_1	N	r_2	p_2	μ_2
.1	.01	1.0	10	.1	.01	1.1

Perf.meas.	Simulation	Analytic	Verdict
P	0.5392 ± 0.0001	0.5391	pass
$\bar{x}$	8.6440 ± 0.0036	8.6425	pass
p_b	0.4070 ± 0.0003	0.4070	pass
p_s	0.0197 ± 0.0001	0.0198	pass
p_{i1}	0.5392 ± 0.0001	0.5391	pass
p_{i2}	0.0000 ± 0.0000	0.0000	pass
p_{o1}	0.0211 ± 0.0001	0.0210	pass
p_{o2}	0.4710 ± 0.0001	0.4710	pass
$\mathbf{p}(0,0,1)$	0.0178 ± 0.0001	0.0179	pass
$\mathbf{p}(0,1,1)$	0.0211 ± 0.0001	0.0210	pass
$\mathbf{p}(N,1,0)$	0.4070 ± 0.0003	0.4070	pass
$\mathbf{p}(N,1,1)$	0.0000 ± 0.0000	0.0000	pass

Table 4.5: Case 3: Finite buffer, $\mu_1 < \mu_2$

r_1	p_1	μ_1	N	r_2	p_2	μ_2
.1	.1	1.0	∞	.1	.01	1.0

Perf.meas.	Simulation	Analytic	Verdict
P	0.5001 $\pm$ 0.0003	0.5000	pass
$\bar{x}$	1.1174 $\pm$ 0.0079	1.1111	pass
p_s	0.4498 $\pm$ 0.0003	0.4500	pass
p_{i1}	0.5001 $\pm$ 0.0003	0.5000	pass
p_{i2}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o1}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o2}	0.5001 $\pm$ 0.0003	0.5000	pass
$\mathbf{p}(0,0,1)$	0.4498 $\pm$ 0.0003	0.4500	pass
$\mathbf{p}(0,1,1)$	0.4090 $\pm$ 0.0004	0.4091	pass

Table 4.6: Case 4: Infinite buffer, $\mu_2 = \mu_2$

r_1	p_1	μ_1	N	r_2	p_2	μ_2
.1	.1	1.1	∞	.1	.01	1.0

Perf.meas.	Simulation	Analytic	Verdict
P	0.5498 $\pm$ 0.0002	0.5500	fail
$\bar{x}$	2.2293 $\pm$ 0.0050	2.2279	pass
p_s	0.3951 $\pm$ 0.0003	0.3950	pass
p_{i1}	0.4998 $\pm$ 0.0002	0.5000	fail
p_{i2}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o1}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o2}	0.5498 $\pm$ 0.0002	0.5500	fail
$\mathbf{p}(0,0,1)$	0.3951 $\pm$ 0.0003	0.3950	pass
$\mathbf{p}(0,1,1)$	0.0000 $\pm$ 0.0000	0.0000	pass

Table 4.7: Case 5: Infinite buffer, $\mu_1 > \mu_2$

r_1	p_1	μ_1	N	r_2	p_2	μ_2
.1	.1	1.0	∞	.1	.01	1.1

Perf.meas.	Simulation	Analytic	Verdict
P	0.5001 $\pm$ 0.0002	0.5000	pass
$\bar{x}$	0.8431 $\pm$ 0.0042	0.8435	pass
p_s	0.4999 $\pm$ 0.0002	0.5000	pass
p_{i1}	0.5001 $\pm$ 0.0002	0.5000	pass
p_{i2}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o1}	0.4279 $\pm$ 0.0003	0.4278	pass
p_{o2}	0.0657 $\pm$ 0.0002	0.0656	pass
$\mathbf{p}(0,0,1)$	0.4610 $\pm$ 0.0002	0.4611	pass
$\mathbf{p}(0,1,1)$	0.4279 $\pm$ 0.0003	0.4278	pass

Table 4.8: Case 5: Infinite buffer, $\mu_1 < \mu_2$

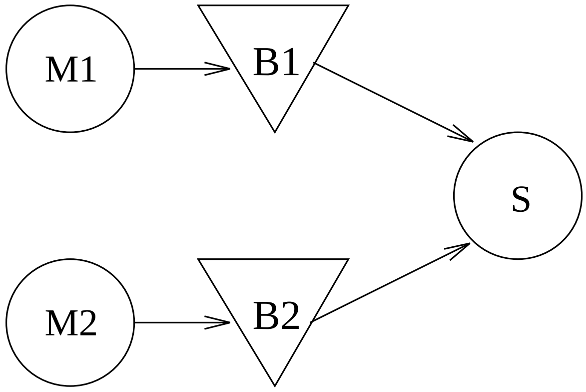

Figure 4-4: A synchronization stage

4.3 Synchronization stage

One central component of our decompositions is a pure synchronization operation that joins one unit of material with one unit of demand. If no demand is present, the material waits, and *vice versa*. As shown in figure 4-4, we model this as a three-machine system. Machines 1 and 2 are unreliable machines of the same type as previously considered, while machine S is fast and reliable. We use machine 1 to represent the production system and machine 2 to represent the demand process. The parameters that describe the synchronization stage are

r_i, the repair rate of machine i, for $i = 1, 2$.
p_i, the failure rate of machine i, for $i = 1, 2$.
μ_i, the maximal processing rate of machine i, for $i = 1, 2$.
N_i, the capacity of buffer i, for $i = 1, 2$.

The state of the system at time t is described by

$\alpha_i(t)$, the repair state of machine i, $\alpha_i \in \{0,1\}$ for $i = 1,2$.
$x_i(t)$, the level of material in buffer, $x_i \in [0,N]$ for $i = 1,2$.
$u_i(t)$, the instantaneous production rate of machine i, $u_i \in [0,\mu_i]$ for $i = 1,2,S$.

Assume that machines 1 and 2 are never starved, that machine S never fails and is never blocked, and that $\mu_S = \max\{\mu_1,\mu_2\}$, so that machine S never slows down the system. There are two important cases: finite or infinite backlogging of unmet demand. Using machine 2 to represent demand, this corresponds to the cases $N_2 < \infty$ and $N_2 = \infty$. We consider these cases separately.

4.3.1 Finite backlog case: $N_2 < \infty$

Under these assumptions, if there is material in both buffers simultaneously, machine S works continuously until at least one buffer is empty. After that, at most one buffer can have a non-zero level, since neither of machines 1 and 2 can work faster then machine S. We can therefore define a single continuous state variable $y = x_1 - x_2 + N_2$, where $y \in [0, N_1 + N_2]$.

Suppose that $0 < y(t) < N_1 + N_2$ at some time t. Then y changes at rate $\alpha_1\mu_1 - \alpha_2\mu_2$, just as in the tandem case. If $y = N_1 + N_2$, machine 1 slows down to the current working rate of machine 2. Similarly, if $y = 0$, machine 2 must slow down to the current working rate of machine 1.

This makes the probabilistic behavior of the state (y,α_1,α_2) identical to the behavior of the state of a tandem two-machine consisting of machines 1 and 2 separated by a buffer of size $N_1 + N_2$, so we can solve for the impulses $\mathbf{p}(y,\alpha_1,\alpha_2)$ and densities using exactly the same method as in section 4.2.1.

Performance measures

For the synchronization stage, several new performance measures that were not used in a tandem line are necessary. We use this notation:

P, the average production rate of the stage
$\bar{x}_i$, the average level of material in buffer i, for $i = 1,2$
p_{bi}, the fraction of time machine i is blocked, for $i = 1,2$
p_s, the fraction of time machine S is starved
p_{si}, the fraction of time machine S is starved due to machine i, for $i = 1,2$
p_{i1}, the fraction of time machine 1 is working at rate μ_1
p_{i2}, the fraction of time machine 1 is working at rate μ_2
p_{o1}, the fraction of time machine S is working at rate μ_1
p_{o2}, the fraction of time machine S is working at rate μ_2
d_s, the fraction of demand that is served from stock
d_b, the fraction of demand that is backlogged
d_r, the fraction of demand that is rejected

We consider machine S *starved due to machine 1* if machine S is working at less than rate μ_S and there is demand, but no finished goods. For example, this is the case if machine 1 is down and there is no finished goods, but some backlogged demand. Or, if $\mu_1 < \mu_2$

y	α_1	α_2	u_1	u_2
N	1	1	$\min\{\mu_1, \mu_2\}$	μ_2
$N_2 < y < N$	1	1	μ_1	μ_2
$0 < y < N_2$	1	1	μ_1	μ_1
0	1	1	μ_1	μ_1
N	1	0	0	0
$N_2 < y < N$	1	0	μ_1	0
$0 < y < N_2$	1	0	μ_1	μ_1
0	1	0	μ_1	μ_1
N	0	1	0	μ_2
$N_2 < y < N$	0	1	0	μ_2
$0 < y < N_2$	0	1	0	0
0	0	1	0	0
N	0	0	0	0
$N_2 < y < N$	0	0	0	0
$0 < y < N_2$	0	0	0	0
0	0	0	0	0

Table 4.9: Input and output rates of the synchronization stage

and $N_2 < \infty$, both machines are up, $x_1 = 0$, and $x_2 = N_2$, the demand process slows down, rejecting some customers, and the synchronization machine is partially starved due to machine 1. Similar statements can be made about situations where machine S is starved due to machine 2.

Finite backlogging

These performance measures can be calculated from the distribution as follows, considering cases with finite backlogging first. We use the state variable $y = x_1 - x_2 + N_2$ to measure the inventory position. For notational convenience, we define $N = N_1 + N_2$, so that $\mathbf{p}(N, 1, 1)$, say, refers to the state where $y = N_1 + N_2$, and both machines are up.

With reference to table 4.9 and the boundary conditions for the tandem line, we have

$$
\begin{aligned}
P &= \mu_2 \frac{r_2}{r_2 + p_2}(1 - p_{b2}) \\
&= \mu_1 \frac{r_1}{r_1 + p_1}(1 - p_{b1}) \\
&= \max\{\mu_1, \mu_2\}(1 - p_s) \\
\bar{x}_1 &= N_1(\mathbf{p}(N, 1, 1) + \mathbf{p}(N, 1, 0)) + \int_{N_2}^{N} (y - N_2) \sum_{\alpha_1=0}^{1} \sum_{\alpha_2=0}^{1} f(y, \alpha_1, \alpha_2)\, dy \\
\bar{x}_2 &= N_2(\mathbf{p}(0, 1, 1) + \mathbf{p}(0, 1, 0)) + \int_{0}^{N_2} (N_2 - y) \sum_{\alpha_1=0}^{1} \sum_{\alpha_2=0}^{1} f(y, \alpha_1, \alpha_2)\, dy
\end{aligned}
$$

$$
\begin{aligned}
p_{b1} &= \mathbf{p}(N,1,0) + (1 - \frac{\mu_2}{\mu_1})\, \mathbf{p}(N,1,1) \\
p_{b2} &= \mathbf{p}(0,0,1) + (1 - \frac{\mu_1}{\mu_2})\, \mathbf{p}(0,1,1) \\
p_{i1} &= \mathbf{p}(0,1,1) + \int_0^N f(y,1,1) + f(y,1,0)\, dy \\
p_{i2} &= \mathbf{p}(N,1,1) \\
p_{o1} &= \mathbf{p}(0,1,1) + \int_0^{N_2} f(y,1,1) + f(y,1,0)\, dy \\
p_{o2} &= \mathbf{p}(N,1,1) + \int_{N_2}^N f(y,1,1) + f(y,0,1)\, dy
\end{aligned}
$$

As a special case, if $\mu_1 = \mu_2$, we have

$$
\begin{aligned}
p_{i1} &= \mathbf{p}(0,1,1) + \mathbf{p}(N,1,1) + \int_0^N f(y,1,1) + f(y,1,0)\, dy \\
p_{i2} &= 0 \\
p_{o1} &= 0 \\
p_{o2} &= \mathbf{p}(N,1,1) + \mathbf{p}(0,1,1) \\
&\quad + \int_0^{N_2} f(y,1,1) + f(y,1,0)\, dy + \int_{N_2}^N f(y,1,1) + f(y,0,1)\, dy
\end{aligned}
$$

We now need to find the starvation probabilities. Let p'_{si} be the probability that the synchronization machine is completely starved due to machine i, causing the flow out of the system to stop completely. Then

$$
\begin{aligned}
p'_{s1} &= \mathbf{p}(0,0,1) + \int_0^{N_2} f(y,0,1) + f(y,0,0)\, dy \\
p'_{s2} &= \mathbf{p}(N,1,0) + \int_{N_2}^N f(y,1,0) + f(y,0,0)\, dy
\end{aligned}
$$

In addition, there is the possibility of partial starvation, where a machine slows down to the processing rate of the other machine. Conceptually, we view this as the machine switching infinitely fast between producing at its own maximal production rate and not producing at all, in such a way that the average production rate over a short interval equals the processing rate of the other machine. This adds a starvation term consisting of the fraction of time spent in this slowed-down state, multiplied by the fraction of time where the production rate needs to be zero in order to match the processing rate of the other machine. There are three cases. If $\mu_1 < \mu_2$,

$$
\begin{aligned}
p_{s1} &= p'_{s1} + \left(1 - \frac{\mu_1}{\mu_2}\right) \left(\mathbf{p}(0,1,1) + \int_0^{N_2} (f(y,1,1) + f(y,1,0))\, dy\right) \\
p_{s2} &= p'_{s2}
\end{aligned}
$$

If $\mu_1 = \mu_2$

$$\begin{aligned} p_{s1} &= p'_{s1} \\ p_{s2} &= p'_{s2} \end{aligned}$$

And, finally, if $\mu_1 > \mu_2$

$$\begin{aligned} p_{s1} &= p'_{s1} \\ p_{s2} &= p'_{s2} + \left(1 - \frac{\mu_2}{\mu_1}\right) \left(\mathbf{p}(N,1,1) + \int_{N_2}^{N} (f(y,1,1) + f(y,0,1))\, dy\right) \end{aligned}$$

In all cases,

$$p_s = p_{s1} + p_{s2}$$

The reason why terms like $f(y,0,1)$ contribute to partial starvations due to machine 2 is that there are finished goods available ($N_2 < y < N$), even though the faster machine 1 is down. Then the synchronization stage is slowed down from its maximal rate $\mu_S = \mu_1$ to the slower rate μ_2. This counts as a partial starvation due to slowness of demand.

The last set of performance measures to be calculated describe how demand is disposed of; whether it is served immediately from stock, or it has to wait first, or it is rejected completely. Since we model the demand process as an unreliable machine, the fractions of arriving demand that gets each treatment are not simply the time averages of the corresponding buffer levels. We must also condition on machine 2 being up, so that demand is in fact arriving. Since conditional probabilities can be calculated as

$$P\{A|B\} = \frac{P\{A \,\&\, B\}}{P\{B\}}$$

where A and B are arbitrary events, with $P\{B\} \neq 0$, we have that the probability of arriving demand being served from stock is

$$\begin{aligned} d_s &= P\{\text{there is stock} \,|\, \text{machine 2 is up}\} \\ &= \frac{P\{\text{there is stock} \,\&\, \text{machine 2 is up}\}}{P\{\text{machine 2 is up}\}} \end{aligned}$$

where the probability that machine 2 is up is

$$P\{\text{machine 2 is up}\} = \mathbf{p}(N,1,1) + \mathbf{p}(0,1,1) + \mathbf{p}(0,0,1) + \int_0^N (f(y,1,1) + f(y,0,1))\, dy$$

and

$$P\{\text{there is stock} \,\&\, \text{machine 2 is up}\} = \mathbf{p}(N,1,1) + \int_{N_2}^{N} (f(y,1,1) + f(y,0,1))\, dy$$

Similarly, the probability that arriving demand is backlogged is

$$d_b = P\{\text{demand must wait} \,|\, \text{machine 2 is up}\}$$

$$= \frac{P\{\text{demand must wait \& machine 2 is up}\}}{P\{\text{machine 2 is up}\}}$$

where

$$P\{\text{demand must wait \& machine 2 is up}\} = \frac{\mu_2}{\mu_1}\mathbf{p}(0,1,1) + \int_0^{N_2} (f(y,1,1) + f(y,0,1))\, dy$$

We have here corrected for the reduced arrival rate when the system is at its maximal backlog. Recall that if $\mu_1 < \mu_2$, $\mathbf{p}(0,1,1) = 0$. Again, we picture the demand machine as switching infinitely fast between a mode where demand is backlogged and a mode where demand is rejected, spending the appropriate fraction of time in each mode.

Finally, demand can be rejected if the finite backlog buffer is full. We have

$$\begin{aligned} d_r &= P\{\text{demand is rejected} \mid \text{machine 2 is up}\} \\ &= 1 - d_s - d_b \end{aligned}$$

4.3.2 Infinite backlog case: $N_2 = \infty$

If unlimited backlogging is allowed, the state variable y as defined in section 4.3.1 ranges from $-\infty$ to $N_1 + N_2$. Negative buffer levels are not defined in the tandem two-machine line, so we cannot use that solution directly for this case.

On the other hand, we can note the amount of free space in this virtual buffer. That is the amount of material that is needed to instantly satisfy all backlogged demand and fill up the finished goods buffer. Call this quantity $y'(t) = N_1 - x_1 + x_2$. We see that $y'(t)$ can range from 0 to ∞, and that it changes at rate $\alpha_2(t)\mu_2 - \alpha_1(t)\mu_1$ when $y'(t) \neq 0$. When $y'(t) = 0$, machine 1 must slow down to the current production rate of machine 2. In fact, $y'(t)$ behaves exactly like the buffer level in a tandem two-machine line consisting of machine 2 followed by machine 1, separated by an infinite buffer.

Since we know how to get the steady state probability distribution of a tandem line with an infinite buffer (section 4.2.2), we also know how to find the steady state distribution of (y', α_2, α_1). We can now calculate the performance measures of interest.

Performance measures

When unlimited backlogging is permitted, stability again becomes an issue. The system is stable if

$$\frac{\mu_2 r_2}{r_2 + p_2} < \frac{\mu_1 r_1}{r_1 + p_1}$$

and we assume that this holds in the following.

We use the state (y', α_2, α_1) to show the relationship to the tandem line more clearly, so in this section, a state $(0,0,1)$ has a full finished goods buffer ($x_1 = N_1$), machine 1 is up, and machine 2 is down. In the notation of the previous section, where the state variable y was used, this state was referred to as $(N,1,0)$. We can now get get the impulses $\mathbf{p}(0, \alpha_2, \alpha_1)$ and the densities $f(y', \alpha_2, \alpha_1)$ as discussed in section 4.2.2.

We calculate the same performance measures as in the finite buffer case. These are

$$
\begin{aligned}
P &= \mu_2 \frac{r_2}{r_2 + p_2} && (4.4)\\
\bar{x}_1 &= N_1(\mathbf{p}(0,0,1) + \mathbf{p}(0,1,1)) + \int_0^{N_1} (N_1 - y') \sum_{\alpha_2=0}^{1} \sum_{\alpha_1=0}^{1} f(y', \alpha_2, \alpha_1)\, dy' && (4.5)\\
\bar{x}_2 &= \int_{N_1}^{\infty} (y' - N_1) \sum_{\alpha_2=0}^{1} \sum_{\alpha_1=0}^{1} f(y', \alpha_2, \alpha_1)\, dy' && (4.6)\\
p_{b1} &= \mathbf{p}(0,0,1) + \left(1 - \frac{\mu_2}{\mu_1}\right) \mathbf{p}(0,1,1) && (4.7)\\
p_{b2} &= 0 && (4.8)\\
p_{s1} &= \int_{N_1}^{\infty} (f(y',0,0) + f(y',1,0))\, dy' && (4.9)\\
p_{s2} &= p_{b1} + \int_0^{N_1} (f(y',0,0) + f(y',1,0))\, dy' && (4.10)\\
p_s &= p_{s1} + p_{s2} && (4.11)\\
p_{i1} &= \int_0^{\infty} (f(y',1,1) + f(y',0,1))\, dy' && (4.12)\\
p_{i2} &= \mathbf{p}(0,1,1) && (4.13)\\
p_{o1} &= \int_{N_1}^{\infty} (f(y',1,1) + f(y',0,1))\, dy' && (4.14)\\
p_{o2} &= \mathbf{p}(0,1,1) + \int_0^{N_1} (f(y',1,1) + f(y',1,0))\, dy' && (4.15)\\
d_s &= \frac{\mathbf{p}(0,1,1) + \int_0^{N_1} (f(y',1,1) + f(y',0,1))\, dy'}{P\{\text{Machine 2 is up}\}} && (4.16)\\
d_b &= \frac{\int_{N_1}^{\infty} (f(y',1,1) + f(y',1,0))\, dy'}{P\{\text{Machine 2 is up}} && (4.17)\\
d_r &= 0 && (4.18)
\end{aligned}
$$

where

$$P\{\text{Machine 2 is up}\} = \mathbf{p}(0,1,1) + \int_0^{\infty} (f(y',1,1) + f(y',1,0))\, dy$$

4.3.3 Validation

As for the tandem line, we have built error catching code into the program implementing these equations, and tested it extensively to ensure that the different cases fit together seamlessly. And again, we have done extensive simulations of the six main cases, as summarized in tables 4.10 to 4.16. All performance measures calculated fell within the 95% confidence intervals in this set of experiments.

r_1	p_1	μ_1	N_1	r_2	p_2	μ_2	N_2
.1	.01	1.0	10	.1	.01	1.0	10

Perf.meas.	Simulation	Analytic	Verdict
P	0.8714 ± 0.0002	0.8714	pass
$\bar{x}_1$	3.7441 ± 0.0168	3.7448	pass
$\bar{x}_2$	3.7448 ± 0.0157	3.7448	pass
p_{b1}	0.0414 ± 0.0002	0.0415	pass
p_{b2}	0.0415 ± 0.0003	0.0415	pass
p_{s1}	0.0643 ± 0.0003	0.0643	pass
p_{s2}	0.0643 ± 0.0002	0.0643	pass
p_{i1}	0.8714 ± 0.0002	0.8714	pass
p_{i2}	0.0000 ± 0.0000	0.0000	pass
p_{o1}	0.0000 ± 0.0000	0.0000	pass
p_{o2}	0.8714 ± 0.0002	0.8714	pass
$\mathbf{p}(N,1,1)$	0.2074 ± 0.0012	0.2075	pass
$\mathbf{p}(N,1,0)$	0.0414 ± 0.0002	0.0415	pass
$\mathbf{p}(0,1,1)$	0.2073 ± 0.0012	0.2075	pass
$\mathbf{p}(0,0,1)$	0.0415 ± 0.0003	0.0415	pass
d_s	0.4772 ± 0.0019	0.4773	pass
d_b	0.4774 ± 0.0017	0.4773	pass
d_r	0.0454 ± 0.0003	0.0454	pass

Table 4.10: Case 1a: Finite backlog, identical machines

r_1	p_1	μ_1	N_1	r_2	p_2	μ_2	N_2
.1	.01	1.0	10	.1	.1	1.0	10

Perf.meas.	Simulation	Analytic	Verdict
P	0.4983 ± 0.0001	0.4982	pass
$\bar{x}_1$	9.2410 ± 0.0036	9.2417	pass
$\bar{x}_2$	0.1824 ± 0.0019	0.1819	pass
p_{b1}	0.4519 ± 0.0003	0.4520	pass
p_{b2}	0.0036 ± 0.0001	0.0036	pass
p_{s1}	0.0138 ± 0.0001	0.0137	pass
p_{s2}	0.4880 ± 0.0002	0.4881	pass
p_{i1}	0.4983 ± 0.0001	0.4982	pass
p_{i2}	0.0000 ± 0.0000	0.0000	pass
p_{o1}	0.0000 ± 0.0000	0.0000	pass
p_{o2}	0.4983 ± 0.0001	0.4982	pass
$\mathbf{p}(N,1,1)$	0.4109 ± 0.0003	0.4109	pass
$\mathbf{p}(N,1,0)$	0.4519 ± 0.0003	0.4520	pass
$\mathbf{p}(0,1,1)$	0.0033 ± 0.0000	0.0032	pass
$\mathbf{p}(0,0,1)$	0.0036 ± 0.0001	0.0036	pass
d_s	0.9495 ± 0.0004	0.9496	pass
d_b	0.0433 ± 0.0003	0.0433	pass
d_r	0.0072 ± 0.0001	0.0071	pass

Table 4.11: Case 1b: Finite backlog, $\mu_1 = \mu_2$, machines not identical

r_1	p_1	μ_1	N_1	r_2	p_2	μ_2	N_2
.1	.01	1.1	10	.1	.1	1.0	10

Perf.meas.	Simulation	Analytic	Verdict
P	0.4986 ± 0.0001	0.4985	pass
$\bar{x}_1$	9.4127 ± 0.0035	9.4106	pass
$\bar{x}_2$	0.1332 ± 0.0017	0.1344	pass
p_{b1}	0.5016 ± 0.0003	0.5015	pass
p_{b2}	0.0030 ± 0.0001	0.0030	pass
p_{s1}	0.0119 ± 0.0001	0.0120	pass
p_{s2}	0.5349 ± 0.0002	0.5349	pass
p_{i1}	0.0629 ± 0.0003	0.0631	pass
p_{i2}	0.4293 ± 0.0003	0.4291	pass
p_{o1}	0.0148 ± 0.0001	0.0149	pass
p_{o2}	0.4822 ± 0.0002	0.4821	pass
$\mathbf{p}(N,1,1)$	0.4293 ± 0.0003	0.4291	pass
$\mathbf{p}(N,1,0)$	0.4625 ± 0.0002	0.4625	pass
$\mathbf{p}(0,1,1)$	0.0000 ± 0.0000	0.0000	pass
$\mathbf{p}(0,0,1)$	0.0030 ± 0.0001	0.0030	pass
d_s	0.9616 ± 0.0004	0.9613	pass
d_b	0.0325 ± 0.0003	0.0327	pass
d_r	0.0059 ± 0.0001	0.0060	pass

Table 4.12: Case 2: Finite backlog, $\mu_1 > \mu_2$

r_1	p_1	μ_1	N_1	r_2	p_2	μ_2	N_2
.1	.01	1.0	10	.1	.1	1.1	10

Perf.meas.	Simulation	Analytic	Verdict
P	0.5465 $\pm$ 0.0002	0.5464	pass
$\bar{x}_1$	8.4721 $\pm$ 0.0041	8.4724	pass
$\bar{x}_2$	0.3128 $\pm$ 0.0024	0.3131	pass
p_{b1}	0.3989 $\pm$ 0.0003	0.3990	pass
p_{b2}	0.0065 $\pm$ 0.0001	0.0065	pass
p_{s1}	0.0228 $\pm$ 0.0001	0.0228	pass
p_{s2}	0.4804 $\pm$ 0.0002	0.4805	pass
p_{i1}	0.5465 $\pm$ 0.0002	0.5464	pass
p_{i2}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o1}	0.0385 $\pm$ 0.0002	0.0385	pass
p_{o2}	0.4618 $\pm$ 0.0002	0.4617	pass
$\mathbf{p}(N,1,1)$	0.0000 $\pm$ 0.0000	0.0000	pass
$\mathbf{p}(N,1,0)$	0.3989 $\pm$ 0.0003	0.3990	pass
$\mathbf{p}(0,1,1)$	0.0069 $\pm$ 0.0001	0.0069	pass
$\mathbf{p}(0,0,1)$	0.0059 $\pm$ 0.0001	0.0059	pass
d_s	0.9175 $\pm$ 0.0005	0.9174	pass
d_b	0.0696 $\pm$ 0.0004	0.0696	pass
d_r	0.0129 $\pm$ 0.0001	0.0130	pass

Table 4.13: Case 3: Finite backlog, $\mu_1 < \mu_2$

r_1	p_1	μ_1	N_1	r_2	p_2	μ_2	N_2
.1	.01	1.0	10	.1	.1	1.0	∞

Perf.meas.	Simulation	Analytic	Verdict
P	0.4999 ± 0.0002	0.5000	pass
$\bar{x}_1$	9.2017 ± 0.0033	9.2015	pass
$\bar{x}_2$	0.3121 ± 0.0035	0.3126	pass
p_{b1}	0.4501 ± 0.0004	0.4500	pass
p_{s1}	0.0141 ± 0.0001	0.0141	pass
p_{s2}	0.4860 ± 0.0003	0.4859	pass
p_{i1}	0.4999 ± 0.0002	0.5000	pass
p_{i2}	0.0000 ± 0.0000	0.0000	pass
p_{o1}	0.0000 ± 0.0000	0.0000	pass
p_{o2}	0.4999 ± 0.0002	0.5000	pass
$\mathbf{p}(N,1,1)$	0.4090 ± 0.0003	0.4091	pass
$\mathbf{p}(N,1,0)$	0.4501 ± 0.0004	0.4500	pass
d_s	0.9488 ± 0.0004	0.9488	pass
d_b	0.0512 ± 0.0004	0.0512	pass

Table 4.14: Case 4: Infinite backlog, $\mu_1 = \mu_2$

r_1	p_1	μ_1	N_1	r_2	p_2	μ_2	N_2
.1	.01	1.1	10	.1	.1	1.0	∞

Perf.meas.	Simulation	Analytic	Verdict
P	0.5000 ± 0.0003	0.5000	pass
$\bar{x}_1$	9.3815 ± 0.0036	9.3826	pass
$\bar{x}_2$	0.2276 ± 0.0034	0.2261	pass
p_{b1}	0.4999 ± 0.0004	0.5000	pass
p_{s1}	0.0122 ± 0.0001	0.0122	pass
p_{s2}	0.5332 ± 0.0003	0.5333	pass
p_{i1}	0.0656 ± 0.0003	0.0656	pass
p_{i2}	0.4278 ± 0.0003	0.4278	pass
p_{o1}	0.0176 ± 0.0001	0.0176	pass
p_{o2}	0.4806 ± 0.0003	0.4807	pass
$\mathbf{p}(N,1,1)$	0.4278 ± 0.0003	0.4278	pass
$\mathbf{p}(N,1,0)$	0.4610 ± 0.0004	0.4611	pass
d_s	0.9612 ± 0.0003	0.9613	pass
d_b	0.0388 ± 0.0003	0.0387	pass

Table 4.15: Case 5: Infinite backlog, $\mu_1 > \mu_2$

r_1	p_1	μ_1	N_1	r_2	p_2	μ_2	N_2
.1	.01	1.0	10	.1	.1	1.1	∞

Perf.meas.	Simulation	Analytic	Verdict
P	0.5499 $\pm$ 0.0003	0.5500	pass
$\bar{x}_1$	8.3880 $\pm$ 0.0051	8.3886	pass
$\bar{x}_2$	0.6122 $\pm$ 0.0061	0.6165	pass
p_{b1}	0.3950 $\pm$ 0.0004	0.3950	pass
p_{s1}	0.0243 $\pm$ 0.0002	0.0243	pass
p_{s2}	0.4758 $\pm$ 0.0003	0.4757	pass
p_{i1}	0.5499 $\pm$ 0.0003	0.5500	pass
p_{i2}	0.0000 $\pm$ 0.0000	0.0000	pass
p_{o1}	0.0471 $\pm$ 0.0003	0.0471	pass
p_{o2}	0.4571 $\pm$ 0.0002	0.4572	pass
$\mathbf{p}(N,1,1)$	0.0000 $\pm$ 0.0000	0.0000	pass
$\mathbf{p}(N,1,0)$	0.3950 $\pm$ 0.0004	0.3950	pass
d_s	0.9143 $\pm$ 0.0005	0.9143	pass
d_b	0.0857 $\pm$ 0.0005	0.0857	pass

Table 4.16: Case 6: Infinite backlog, $\mu_1 < \mu_2$

Chapter 5

Decomposition equations

Now that we have developed a kit of building blocks, we investigate how the building blocks affect each other when joined. Recall that, in a decomposition, each "machine" represents the behavior of the portion of the system on one side of a particular buffer, so we must consider the flow pattern into and out of the buffers.

5.1 Tandem equations

As noted in section 1.1.2, there are several existing methods for analyzing an inhomogeneous continuous material production line. All of these methods have both advantages and disadvantages: The method of Glassey and Hong (1986) describes a simplified model, and the resulting equations are hard to solve. The approach by Semery (1987) is fast and simple, but is not very accurate. The one by Alvarez Vargas, Dallery, and David (1991a) has convergence problems. And finally, the method developed by Burman (1995) is fast, reliable, and reasonably accurate, but the derivation of the equations is complicated and involves some non-obvious substitutions of variables to achieve good performance. It is therefore difficult to extend to other systems.

We now develop yet another set of decomposition equations for an inhomogeneous tandem line of unreliable machines processing continuous material. Our aim is to come up with a simple method that gives reasonably good results.

Our notation refers to parameters that belong to a particular line in the decomposition by the line index in parentheses, and uses subscripts that indicate whether this is the upstream or downstream machine in that line. For instance, $r_u(i)$ is the repair rate of the upstream machine in two-machine line i. This should not be confused with r_i, the repair rate of the corresponding machine in the original line.

The unknowns of the decomposition are the parameters of the machines in each line, which are

$\mu_u(i)$, the upstream processing rate, for all lines $i = 1, \ldots, k-1$
$r_u(i)$, the upstream repair rate, for all lines $i = 1, \ldots, k-1$
$p_u(i)$, the upstream failure rate, for all lines $i = 1, \ldots, k-1$
$\mu_d(i)$, the downstream processing rate, for all lines $i = 1, \ldots, k-1$
$r_d(i)$, the downstream repair rate, for all lines $i = 1, \ldots, k-1$

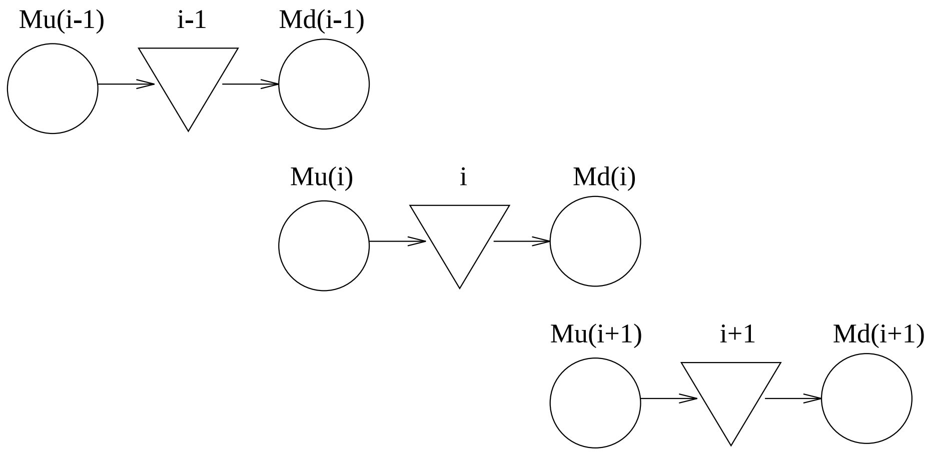

Figure 5-1: Tandem two-machine lines in a decomposition

$p_d(i)$, the downstream failure rate, for all lines $i = 1, \ldots, k-1$

For a k-machine line, this gives $6(k-1)$ unknowns, since the unknowns are associated with the buffers between the machines. Since we assume that the first machine in a tandem line is never starved and the last machine never blocked, the machines at the ends of the decomposition also just represent the behavior of these machines. As a consequence

$$\begin{aligned}
\mu_u(1) &= \mu_1 && (5.1)\\
r_u(1) &= r_1 && (5.2)\\
p_u(1) &= p_1 && (5.3)\\
\mu_d(k-1) &= \mu_k && (5.4)\\
r_d(k-1) &= r_k && (5.5)\\
p_d(k-1) &= p_k && (5.6)
\end{aligned}$$

This leaves $6(k-2)$ unknowns to be determined.

5.1.1 Processing rates

We first derive a new set of equations that determine the processing rates $\mu_d(i)$ and $\mu_u(i)$ for each two-machine line i in the inhomogeneous decomposition.

Consider buffer i in the original line. The flow into this buffer occasionally arrives at rate μ_i, but sometimes at other rates corresponding to slower machines upstream (if any), or it may be stopped altogether. In the decomposition, the behavior when material is flowing into the buffer must be captured in the parameter $\mu_u(i)$. Similarly, whatever is going on upstream of buffer $i-1$ is captured in the description of machine $M_u(i-1)$. We can recursively describe the flow into buffer i in terms of $\mu_u(i-1)$ and μ_i, given that the flow is non-zero.

We can do this by using the performance measures p_{o1} and p_{o2} we developed in the previous section. Referring to the decomposed lines, $p_{o1}(i-1)$ is the probability of seing output from line $i-1$ at rate $\mu_u(i-1)$, and $p_{o2}(i-1)$ is the probability of seeing output at rate $\mu_d(i-1)$, as evaluated by the two-machine solution. But, in two-machine line $i-1$, $\mu_d(i-1)$ is used to describe how the downstream portion of the line affects the flow out of buffer $i-1$, and does not directly affect the flow into buffer i. We therefore replace $\mu_d(i-1)$ by μ_i, the production rate of the corresponding machine in the original line. Then

$$\mu_u(i) = \frac{p_{o1}(i-1)\mu_u(i-1) + p_{o2}(i-1)\mu_i}{p_{o1}(i-1) + p_{o2}(i-1)} \tag{5.7}$$

where the denominator is the probability of the flow rate being non-zero.

The flow into buffer i can also occur at rates determined by machines *downstream* from buffer i. For instance, if $\mu_{i-1} > \mu_i > \mu_{i+1}$, all three machines are up, and buffers $i-1$ and i are full, the rate is μ_{i+1} (or even lower, depending on what is further downstream). This is not captured in equation (5.7), since we are now concerned with effects that propagate from upstream in the line. In this respect, equation (5.7) falls midway between the methods of Semery (1987) and Alvarez Vargas, Dallery, and David (1991a). In the first case, $\mu_u(i)$ was simply set to equal μ_i for all i. In the second, a more complicated derivation was done to account for effects propagating into the buffer from downstream.

By similar reasoning, we get

$$\mu_d(i) = \frac{p_{i1}(i+1)\mu_{i+1} + p_{i2}(i-1)\mu_d(i+1)}{p_{i1}(i-1) + p_{i2}(i-1)} \tag{5.8}$$

where $p_{i1}(i+1)$ is the fraction of time line $i+1$ has an input rate of $\mu_u(i+1)$, and $p_{i2}(i+1)$ is the fraction of time line $i+1$ has an input rate of $\mu_d(i+1)$.

5.1.2 Flow rate/idle time

We now derive an equation for the ratio between failure and repair rates of machine $M_u(i)$, using the same approach as in Dallery, David, and Xie (1989). The difference is that we now consider an inhomogeneous system, so we must account for different processing rates in the derivation. We define

$$I_u(i) = p_u(i)/r_u(i) \text{ for all } i = 1, 2, \ldots, k-1 \tag{5.9}$$

$$I_d(i) = p_d(i)/r_d(i) \text{ for all } i = 1, 2, \ldots, k-1 \tag{5.10}$$

Our starting points are the flow rate/idle time relationship and the conservation of flow. From Gershwin (1994), we have the flow rate/idle time equations:

$$P(i-1) = \mu_d(i-1)e_d(i-1)(1-p_s(i-1)) \tag{5.11}$$

$$P(i) = \mu_u(i)e_u(i)(1-p_b(i)) \tag{5.12}$$

$$P(i) = \mu_i e_i(1-p_{si}-p_{bi}) \tag{5.13}$$

where

$$e_d(i-1) = \frac{r_d(i-1)}{r_d(i-1)+p_d(i-1)} = \frac{1}{1+I_d(i-1)}$$
$$e_u(i) = \frac{r_u(i)}{r_u(i)+p_u(i)} = \frac{1}{1+I_u(i)}$$

and p_{si} and p_{bi} are the probabilities of starvation and blockage of machine i in the original line, respectively. We can rewrite these equations as

$$\begin{aligned} p_s(i-1) &= 1 - \frac{P(i-1)}{\mu_d(i-1)e_d(i-1)} \\ p_b(i) &= 1 - \frac{P(i)}{\mu_u(i)e_u(i)} \end{aligned}$$

Conservation of flow states that

$$P(1) = P(2) = \cdots = P(k) \tag{5.14}$$

Moreover, we assume that the starvation and blocking probabilities in the decomposed system equals the corresponding quantities in the original line. This merely states that the decomposition procedure we are about to describe will work. Formally,

$$p_{si} = p_s(i-1) \tag{5.15}$$
$$p_{bi} = p_b(i) \tag{5.16}$$

We then have that

$$\begin{aligned} P(i) &= \mu_i e_i(1 - p_{si} - p_{bi}) \\ &= \mu_i e_i(1 - p_s(i-1) - p_b(i)) \\ &= \mu_i e_i \left(1 - 1 + \frac{P(i-1)}{\mu_d(i-1)e_d(i-1)} - 1 + \frac{P(i)}{\mu_u(i)e_u(i)}\right) \\ &= \mu_i e_i \left(\frac{P(i)}{\mu_d(i-1)e_d(i-1)} + \frac{P(i)}{\mu_u(i)e_u(i)} - 1\right) \end{aligned}$$

We have here inserted equations (5.15) and (5.16) in equation (5.13). We then used equations (5.14) and (5.14) to express this in terms of the production rates and the parameters of the two-machine lines, before simplifying. Dividing through by $P(i)$, we have

$$\begin{aligned} 1 &= \mu_i e_i \left(\frac{1}{\mu_d(i-1)e_d(i-1)} + \frac{1}{\mu_u(i)e_u(i)} - \frac{1}{P(i)}\right) \\ &= \mu_i e_i \left(\frac{1}{\mu_d(i-1)}(1 + I_d(i-1)) + \frac{1}{\mu_u(i)}(1 + I_u(i)) - \frac{1}{P(i-1)}\right) \end{aligned}$$

We now used equations (5.14) and (5.14) to express the isolated efficiencies $e_u(i)$ and $e_d(i)$

in terms of $I_u(i)$ and $I_d(i)$. Rearranging terms, we get

$$\begin{aligned}\frac{1}{\mu_i e_i} &= \frac{1}{\mu_d(i-1)}(1+I_d(i-1)) + \frac{1}{\mu_u(i)}(1+I_u(i)) - \frac{1}{P(i-1)} \\ \frac{\mu_u(i)}{\mu_i e_i} &= \frac{\mu_u(i)}{\mu_d(i-1)}(1+I_d(i-1)) + 1 + I_u(i) - \frac{\mu_u(i)}{P(i-1)}\end{aligned}$$

Isolating $I_u(i)$ on the left side, we arrive at

$$I_u(i) = \frac{\mu_u(i)}{\mu_i e_i} + \frac{\mu_u(i)}{P(i-1)} - \frac{\mu_u(i)}{\mu_d(i-1)}(1+I_d(i-1)) - 1 \tag{5.17}$$

If all processing rates μ_i are set to 1.0, the $\mu_u(i)$ and $\mu_d(i)$ also become 1.0 for all i (from equation (5.7 and 5.8)), and this becomes the same as the corresponding equation in Dallery, David, and Xie (1989). The method by Alvarez Vargas, Dallery, and David (1991a) also uses this equation by imposing some extra constraints on the processing rates. Burman (1995) uses a similar equation to fix the processing rate $\mu_u(i)$ after calculating $r_u(i)$ and $p_u(i)$ by other methods.

Through a similar derivation, we get

$$I_d(i) = \frac{\mu_d(i)}{\mu_{i+1} e_{i+1}} + \frac{\mu_d(i)}{P(i+1)} - \frac{\mu_d(i)}{\mu_u(i+1)}(1+I_u(i+1)) - 1 \tag{5.18}$$

5.1.3 Resumption of flow

We now consider the way flow resumes after an interruption to find $r_u(i)$. We will derive this equation by a different route than that followed by Burman (1995), although the result will be the same. Recall that the system is inhomogeneous, so a machine may sometimes be forced to operate at a speed that is lower than its own maximal rate. One way this can occur is by starvation. We distinguish between a partial starvation, where the machine operates at a decreased, but non-zero rate; and a complete starvation, where it cannot operate at all. Partial starvations were accounted for in equation (5.7) which determined the average flow rate $\mu_u(i)$ for the situations where the flow is non-zero. We are now concerned with situations where the flow rate into buffer i becomes zero, either due to a failure of machine i or a complete starvation of machine i.

The average time to resume flow into buffer i is a probabilistically weighted combination of the time it takes given that the interruption is due to a failure of machine i, and the time it takes given that the interruption is due to a complete starvation of machine i. Since we assume exponential repair times, the time to resume flow into the buffer is $1/r_u(i)$ in the decomposition. This must match the average duration of an interruption of flow into buffer i in the original line.

We can then write

$$\frac{1}{r_u(i)} = X_u(i)\bar{S}_i + (1 - X_u(i))\frac{1}{r_i} \tag{5.19}$$

where $\bar{S}_i$ is the average duration of a starvation of machine i, and $X_u(i)$ is the fraction of all interruptions of flow into buffer i that are due to starvations propagating from upstream.

This is the average number of starvations per unit of time, divided by the average number of interruptions per unit of time.

The duration $\bar{S}_i$ of a starvation of machine i is either the repair time of the upstream machine $i-1$, or the duration of a starvation of that machine. Both of these are captured in the decomposition by a failure of the upsteam machine in line $i-1$, and we have

$$\bar{S}_i = \frac{1}{r_u(i-1)} \tag{5.20}$$

We now find the weighting factor $X_u(i)$. Note that the fraction of time where the machine is interrupted is the average duration of an interruption times the average number of interruptions per unit of time. We know that the duration of an interruption of machine $M_u(i)$ is $1/r_u(i)$, that the duration of a failure of machine i is $1/r_i$, and that the fraction of time machine i is completely starved is $p'_s(i-1)$. We have that

$$X_u(i) = \frac{p'_s(i-1)r_u(i-1)}{p_{du}(i)r_u(i)} \tag{5.21}$$

where $p_{du}(i)$ is the fraction of time the upstream machine in line i is down. Since the number of repairs must equal the number of failures, this is

$$\begin{aligned}
p_{du}(i) &= \frac{p_u(i)}{r_u(i)+p_u(i)}(1-p_b(i)) \\
&= (1-e_u(i))(1-p_b(i)) \\
&= (1-p_b(i)) - e_u(i)(1-p_b(i) \\
&= \frac{P(i)}{\mu_u(i)e_u(i)} - \frac{P(i)}{\mu_u(i)} \\
&= \frac{P(i)}{\mu_u(i)}(\frac{1}{e_u(i)}-1) \\
&= \frac{P(i)}{\mu_u(i)}I_u(i)
\end{aligned}$$

The fourth equality in this sequence comes from the flow rate/idle time relationship (5.12). Inserting this result into equation (5.21), we have

$$X_u(i) = \frac{p'_s(i-1)r_u(i-1)\mu_u(i)}{P(i)I_u(i)r_u(i)} = \frac{\mathbf{p}(0,0,1,i-1)r_u(i-1)\mu_u(i)}{P(i)I_u(i)r_u(i)} \tag{5.22}$$

We can now find $r_u(i)$. Substituting equations (5.22) and (5.20) into equation (5.19) and rearranging terms, we have that

$$r_u(i) = \frac{\mathbf{p}(0,0,1,i-1)\mu_u(i)}{P(i)I_u(i)}r_u(i-1) + \left(1 - \frac{\mathbf{p}(0,0,1,i-1)\mu_u(i)}{P(i)I_u(i)}\right)r_i \tag{5.23}$$

This equation is the same as in Burman (1995) (but we derived it in a different way). It differs from the corresponding equation of Dallery, David, and Xie (1989) by using the ratio $P(i)/\mu_u(i)$ instead of the efficiency $E_u(i)$ of the machine, defined as the fraction of time it is working. This is necessary to account for the different processing rates of the machines.

Equation (5.23) together with equation (5.17) also determine the value of $p_u(i)$, since, by the definition of I_u (equation (5.9) we have

$$p_u(i) = I_u(i) r_u(i) \tag{5.24}$$

By similar reasoning, we also get that

$$\begin{aligned} r_d(i) &= \frac{\mathbf{p}(N,1,0,i+1)\mu_d(i)}{P(i+1)I_d(i)} r_d(i+1) \\ &\quad + \left(1 - \frac{\mathbf{p}(N,1,0,i+1)\mu_d(i)}{P(i+1)I_d(i)}\right) r_{i+1} \qquad (5.25) \\ p_d(i) &= I_d(i) r_d(i) \qquad (5.26) \end{aligned}$$

We now have the equations we need to describe the system. One aspect of equation (5.18) must be mentioned. It is derived exactly like equation (5.17), except that an occurrence of $P(i)$ is substituted by $P(i+1)$ instead of the opposite substitution $P(i+1) \rightarrow P(i)$ in the previous case. This is not just for convenience, but is necessary to get the $6(k-2)$ independent equations we need. This idea is due to Dallery, David, and Xie (1988).

5.2 Synchronization stage equations

We assume that the synchronization stage has tandem lines as its immediate upstream and downstream neighbors, if any, as shown in figure 5-2. We assume that the buffer immediately downstream of the synchronization stage is infinitely large. Otherwise, blockages might propagate into the synchronization stage, which could cause the two buffers in the synchronization stage to have positive levels at the same time. This is not allowed by the model we developed in section 4.3. This restriction is quite acceptable in the models we study in this thesis. Under this assumption, the synchronization machine remains fast and reliable as in the isolated synchronization stage of section 4.3.

Notice that machine $M_d(k)$ in the synchronization stage can never be starved, since there are no machines upstream of it. This is used to model the external demand process. A "failure" of the demand process can be any event that interrupts the arrival of demand. The failures and repairs can also be used to match the variability of some other type of demand process with non-constant demand rate. The parameters of the demand "machine" $M_d(k)$ are then fixed at the corresponding values of the demand process. Let

μ_D be the demand arrival rate when the demand process is working
p_D be the failure rate of the demand process
r_D be the repair rate of the demand process

5.2.1 Parameters of the synchronization stage k

The upstream machine $M_u(k)$ in the synchronization stage k represents the entire system upstream of this stage in the decomposition. Its parameters are determined by those of

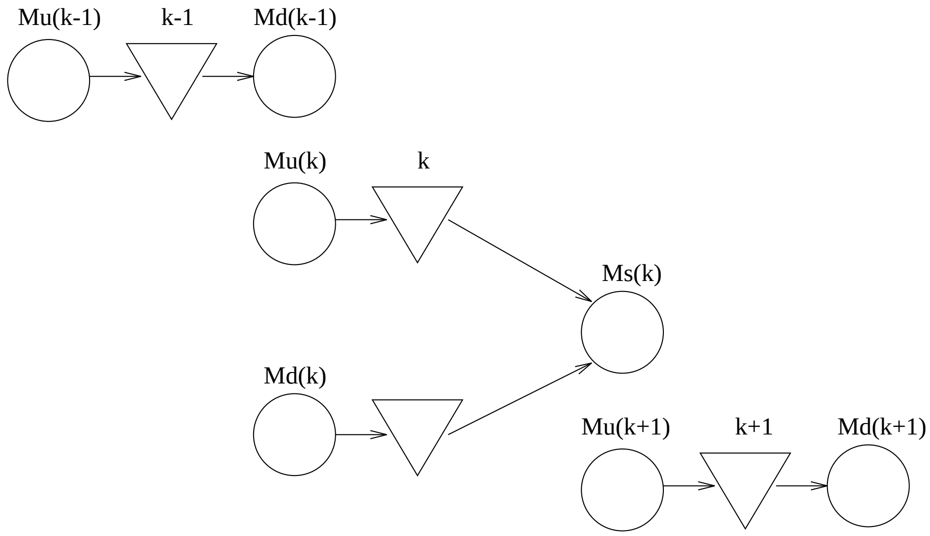

Figure 5-2: Synchronization stage in a decomposition

the next upstream two-machine line and those of the corresponding machine in the original system, just as in a tandem line. This is given by equations (5.17), (5.23), and (5.24).

The downstream machine $M_d(k)$ represents the demand process. This machine is not affected by the rest of the system except through the backlog buffer, which is part of the synchronization stage solution. Its parameters are

$$\mu_d(k) = \mu_D \tag{5.27}$$
$$r_d(k) = r_D \tag{5.28}$$
$$p_d(k) = p_D \tag{5.29}$$

The synchronization machine $M_s(k)$ is assumed to be fast and perfectly reliable. We have also assumed that buffer $k+1$ is infinite, so machine $M_s(k)$ is never blocked. Then its parameters do not change during the decomposition, since no disturbances can propagate into the synchronization stage from downstream. For concreteness, we have

$$\mu_s(k) = \max\{\mu_u(k), \mu_d(k)\} \tag{5.30}$$
$$r_s(k) = \infty \tag{5.31}$$
$$p_s(k) = 0 \tag{5.32}$$

5.2.2 Parameters of the downstream line $k+1$

We now know how the synchronization stage is affected by its neighbors, and must consider the opposite interaction. We need to find the parameters $\mu_d(k-1)$, $r_d(k-1)$, and $p_d(k-1)$ of the downstream machine in the two-machine line upstream from the synchronization stage, and the parameters $\mu_u(k+1)$, $r_u(k+1)$, and $p_u(k+1)$ of the upstream machine in the two-machine line downstream from it.

First consider the upstream machine in the tandem two-machine line $k+1$. This machine is never blocked, since buffer $k+1$ is assumed to be infinitely large. By similar reasoning as for the tandem case,

$$\mu_u(k+1) = \frac{p_{o1}(k)\mu_u(k) + p_{o2}(k)\mu_D}{p_{o1}(k) + p_{o2}(k)} \tag{5.33}$$

where $p_{o1}(k)$ and $p_{o2}(k)$ are the fractions of time the flow rate out of the synchronization stage k are $\mu_u(k)$ and μ_D, respectively.

Now consider interruptions of flow into buffer $k+1$. This can be due to two causes: Either the demand process is stopped, or the production process was interrupted. In the latter case, machine k could have failed, or it could have starved due to something happening upstream. Both of these possibilities are captured in the parameters of machine $M_u(k)$ in the decomposition. From the flow rate/idle time relationship at two-machine line $k+1$, we have

$$\begin{aligned} P(k+1) &= \mu_u(k+1)e_u(k+1)(1-p_b(k+1)) \\ &= \mu_u(k+1)e_u(k+1) \\ &= \mu_u(k+1)\frac{1}{1+I_u(k+1)} \end{aligned}$$

Conservation of flow demands that $P(k) = P(k+1)$. Then

$$\begin{aligned} P(k) &= \frac{\mu_u(k+1)}{1+I_u(k+1)} \\ 1+I_u(k+1) &= \frac{\mu_u(k+1)}{P(k)} \end{aligned}$$

This leads to the expression

$$I_u(k+1) = \frac{\mu_u(k+1)}{P(k)} - 1 \tag{5.34}$$

Recall that in section 4.3 we used the symbols p'_{s1} and p'_{s2} to represent the fractions of time the synchronization stage is completely starved with respect to machines 1 and 2. Now, machine 1 is the upstream machine $M_u(k)$ and machine 2 the demand process $M_d(k)$. Note that by the definitions in section 4.3, the events "synchronization stage completely starved by machine 1" and "synchronization stage completely starved by machine 2" are mutually exclusive. Machine $M_s(k)$, the synchronization operation, is reliable, so if the flow out of the synchronization stage is stopped, this must be due to one of machines 1 or 2. Therefore the fraction of time the flow into buffer $k+1$ is interrupted is

$$p'_s(k) = p'_{s1}(k) + p'_{s2}(k)$$

Similar to the derivation in section 5.1.3, the fraction of all interruptions of flow into buffer $k+1$ that are due to machine $M_u(k)$ is

$$X_u(k) = \frac{p'_{s1}(k)r_u(k)}{p'_s(k)r_u(k+1)}$$

The average duration of one of these disruptions is $1/r_u(k)$, and we have that

$$\begin{aligned}
\frac{1}{r_u(k+1)} &= X_u(k)\frac{1}{r_u(k)} + (1 - X_u(k))\frac{1}{r_D} \\
\frac{1}{r_u(k+1)} &= \frac{p'_{s1}(k)r_u(k)}{p'_s(k)r_u(k+1)}\frac{1}{r_u(k)} + \left(1 - \frac{p'_{s1}(k)r_u(k)}{p'_s(k)r_u(k+1)}\right)\frac{1}{r_D}
\end{aligned}$$

Cancelling and rearranging terms, we have that

$$r_u(k+1) = \frac{p'_{s1}(k)}{p'_s(k)}r_u(k) + \left(1 - \frac{p'_{s1}(k)}{p'_s(k)}\right) r_D \tag{5.35}$$

We obtain $p_u(k+1)$ from the definition of $I_u(k+1)$, equation (5.9), giving

$$p_u(k+1) = I_u(k+1)r_u(k+1) \tag{5.36}$$

5.2.3 Parameters of the upstream line $k-1$

The derivation of the equations for the downstream machine $M_d(k-1)$ in the next upstream two-machine line $k-1$ is similar to that for the tandem line, except that no blockages can propagate into the synchronization stage, so the parameters of the demand machine take the place of the parameters of the next downstream line in the decompositon. Then

$$\begin{aligned}
\mu_d(k-1) &= \frac{p_{i1}(k)\mu_k + p_{i1}(k)\mu_D}{p_{i1}(k) + p_{i2}(k)} &(5.37)\\
I_d(k-1) &= \frac{\mu_d(k-1)}{\mu_k e_k} + \frac{\mu_d(k-1)}{P(k)} - \frac{\mu_d(k-1)}{\mu_u(k)}(1 + I_u(k)) - 1 &(5.38)\\
r_d(k-1) &= \frac{\mathbf{p}(N,1,0,i)\mu_d(k-1)}{P(k)I_d(k-1)}r_D \\
&\quad + \left(1 - \frac{\mathbf{p}(N,1,0,i)\mu_d(k-1)}{P(k)I_d(k-1)}\right) r_k &(5.39)\\
p_d(k-1) &= I_d(k-1)r_d(k-1) &(5.40)
\end{aligned}$$

We now have the decomposition equations we need to start assembling the components to system representations.

Chapter 6

Assembling and solving the models

We have developed several kinds of building blocks for representations of manufacturing systems, and we have derived the equations that link these building blocks together. We now assemble these blocks to representations of manufacturing system, including their control policies. For now, are mainly concerned with tandem lines under various control schemes, but this approach can be extended to other types of systems such as assembly/disassembly systems. Some of our models, such as the closed loop model in section 6.3, can be interpreted as either a control policy of a tandem line (CONWIP in saturating demand, in this case) or a non-tandem system.

In this chapter, we also present iterative algorithms to solve the resulting sets of equations. These are extensions and modifications of the algorithms by Dallery, David, and Xie (1989) and Frein, Commault, and Dallery (1994). We proceed by the type of system.

6.1 Tandem line in saturating demand

6.1.1 Assembling the model

For the tandem line, we use the same approach as Dallery, David, and Xie (1989) and many other papers, but we use the equations derived in section 5.1 of this thesis to connect the components. The behavior of each buffer is represented by a pair of machines. These machines and the buffer constitute a tandem two-machine line, as shown in figure 6-1. The parameters of the upstream machine in each two-machine line is determined by equations (5.7), (5.17), (5.23), and (5.24), which are

$$
\begin{aligned}
\mu_u(i) &= \frac{p_{o1}(i-1)\mu_u(i-1) + p_{o2}(i-1)\mu_i}{p_{o1}(i-1) + p_{o2}(i-1)} \\
I_u(i) &= \frac{\mu_u(i)}{\mu_i e_i} + \frac{\mu_u(i)}{P(i-1)} - \frac{\mu_u(i)}{\mu_d(i-1)}(1 + I_d(i-1)) - 1 \\
r_u(i) &= \frac{\mathbf{p}(0,0,1,i-1)\mu_u(i)}{P(i)I_u(i)} r_u(i-1) + \left(1 - \frac{\mathbf{p}(0,0,1,i-1)\mu_u(i)}{P(i)I_u(i)}\right) r_i \\
p_u(i) &= I_u(i) r_u(i)
\end{aligned}
$$

The parameters of the downstream machine are determined by equations (5.8), (5.18),

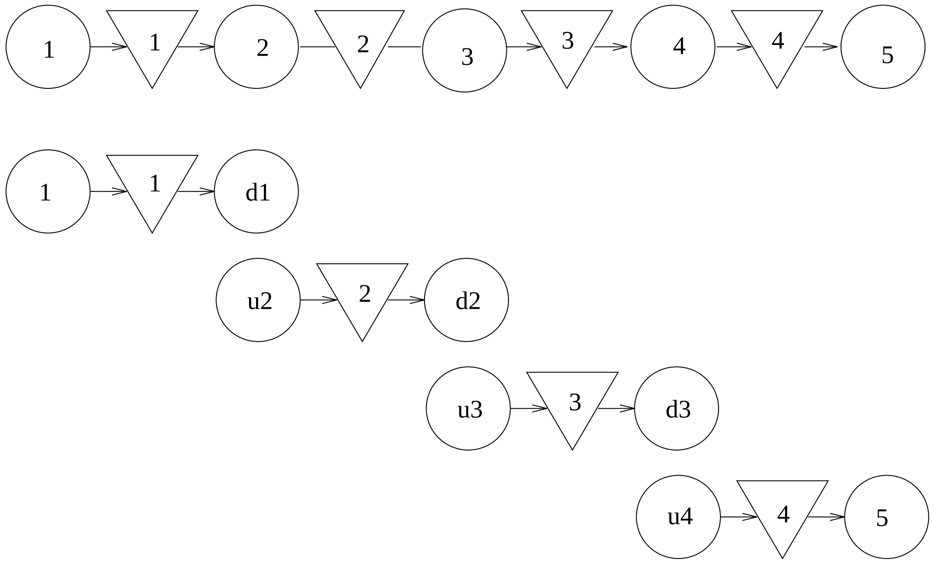

Figure 6-1: Decomposing a tandem line

(5.25), and (5.26). These are

$$\begin{aligned}
\mu_d(i) &= \frac{p_{i1}(i+1)\mu_{i+1} + p_{i2}(i+1)\mu_d(i+1)}{p_{i1}(i+1) + p_{i2}(i+1)} \\
I_d(i) &= \frac{\mu_d(i)}{\mu_{i+1}e_{i+1}} + \frac{\mu_d(i)}{P(i+1)} - \frac{\mu_d(i)}{\mu_u(i+1)}(1 + I_u(i+1)) - 1 \\
r_d(i) &= \frac{\mathbf{p}(N,1,0,i+1)\mu_d(i)}{P(i+1)I_d(i)} r_d(i+1) \\
&\quad + \left(1 - \frac{\mathbf{p}(N,1,0,i+1)\mu_d(i)}{P(i+1)I_d(i)}\right) r_{i+1} \\
p_d(i) &= I_d(i)r_d(i)
\end{aligned}$$

As shown by the figure (and equations (5.1) through (5.6)), the upstream machine of the first line and the downstream machine of the last line represent the first and last machines in the original line, respectively.

6.1.2 Solving the tandem model

The solution algorithm for the tandem model is essentially the same as in every decomposition paper since Dallery, David, and Xie (1988). We initialize the unknown variables ($\mu_u(i)$, $r_u(i)$, $p_u(i)$, $\mu_d(i)$, $r_d(i)$, and $p_d(i)$) for each two-machine line $i \in \{1, k-1\}$ to the corresponding values of the machines in the original line. We then move downstream from the start of the line, evaluating each two-machine line as described in chapter 4. Every time a two-machine line has been evaluated, we propagate the results to the upstream end of the next downstream two-machine line through the equations derived in chapter 5. We then evaluate that two-machine line, and continue doing this until we have reached the end of the system. Then we turn around, and do the same procedure moving upstream, this time propagating results to the downstream machine in the next upstream two-machine line. We do this until we again reach the top of the system, which completes one iteration.

We repeat these iterations until the equations converge. Our measure of convergence is the maximum relative change in any of the unknowns, where the relative change of any variable x is computed as

$$\delta = \left| \frac{x_{\text{new}} - x_{\text{old}}}{x_{\text{old}}} \right| \tag{6.1}$$

The old value is the one computed in the previous iteration. This is the same convergence criterion as used by Dallery, David, and Xie (1989). Other authors, for instance Burman (1995), use conservation of flow as a convergence measure, terminating the algorithm when the difference between the production rate of different two-machine lines is sufficiently small. This algorithm can be extended to general tree-structured networks, see, for instance, Di Mascolo, David, and Dallery (1991) or Gershwin (1991).

We can now state the complete algorithm, and will do this by a series of procedures that together constitute a complete implementation. The description is in a Pascal-like pseudo-code. The keywords of the pseudo-code are shown in boldface, e.g., **begin**. Variable names are shown in italics, e.g., *Diff*. We use ":=" for the assignment operator. We use the "**while** (*condition*)" statement as the only loop control mechanism. It is assumed to execute its body (the **begin** ... **end** block that follows it) when the *condition* evaluates to **true**, and pass control to the statement following the body when the *condition* evaluates to **false**.

First, we state a procedure to update a variable. We specify whether an argument to a procedure should be passed by reference (i.e., a pointer in C, pointer or reference in C++, the default passing convention in FORTRAN) or by value. Passing the argument by reference is necessary when we want a change in the value of the argument to be visible outside the procedure.

```
procedure UpdateVariable(real x: by reference, real v: by value)
begin
    Diff := AbsoluteValue( (x − v)/v );
    MaxDiff := Max( MaxDiff, Diff );
    x := v;
end
```

Here, *MaxDiff* is a global variable that measures the degree of convergence so far. The procedures AbsoluteValue and Max are assumed to be available.

We can now state procedures to evaluate a two-machine line and propagate its behavior downstream and upstream, respectively. We assume that there exists a procedure EvaluateTwoMachineLine that takes the index of a two-machine line in the decomposition as an argument and calculates its performance measures based on the current values of the parameters of that line, as described in section 4.2.

```
procedure PropagateToDownstream(integer i: by value)
begin
    EvaluateTwoMachineLine( i );
    UpdateVariable( μ_u(i + 1), equation (5.7) );
```

```
        Calculate I_u(i + 1) from equation (5.17);
        UpdateVariable( r_u(i + 1), equation (5.23) );
        UpdateVariable( p_u(i + 1), equation (5.24) );
end

procedure PropagateToUpstream(integer i: by value)
begin
        EvaluateTwoMachineLine( i );
        UpdateVariable( μ_d(i − 1), equation (5.8) );
        Calculate I_d(i − 1) from equation (5.18);
        UpdateVariable( r_d(i − 1), equation (5.25) );
        UpdateVariable( p_d(i − 1), equation (5.26) );
end
```

Now, we can state procedures to perform the two halves of an iteration.

```
procedure TandemSweepDownstream
begin
        i := 1;
        while ( i ≤ k − 2 )
        begin
                PropagateToDownstream( i );
                i := i + 1;
        end
end

procedure TandemSweepUpstream
begin
        i := k − 1;
        while ( i ≥ 2 )
        begin
                PropagateToUpstream( i );
                i := i − 1;
        end
end
```

We need a procedure to read system parameters from input, and one to initialize the parameters of each two-machine line from the original system parameters. Recall that k is the number of machines, μ_i, r_i, and p_i parameters of machine i for $i = 1, 2, \ldots, k$, and N_i the size of buffer i for $i = 1, 2, \ldots, k-1$.

```
procedure TandemReadInput
begin
        Read k;
        i := 1;
        while ( i ≤ k )
```

```
        begin
                Read μ_i, r_i, and p_i;
                if ( i < k ) Read N_i;
                endif
        end
end

procedure TandemInitialize
begin
        i := 1;
        while ( i ≤ k − 1 )
        begin
                μ_u(i) := μ_i;
                r_u(i) := r_i;
                p_u(i) := p_i;
                I_u(i) := p_i/r_i;

                μ_d(i) := μ_{i+1};
                r_d(i) := r_{i+1};
                p_d(i) := p_{i+1};
                I_d(i) := p_{i+1}/r_{i+1};

                i := i + 1;
        end
end
```

Finally, we state the complete algorithm.

```
procedure TandemDecomposition
begin
        TandemReadInput;
        TandemInitialize;

        MaxDiff := 1;
        while ( MaxDiff > Epsilon )
        begin
                MaxDiff := 0;
                TandemSweepDownstream;
                TandemSweepUpstream;
        end

        PrintOutput;
end
```

The value *Epsilon* is a sufficiently small value, for instance 10^{-6}. We have here assumed that a procedure PrintOutput exists that generates a properly formatted report on performance measures of interest from the final values of the two-machine line performance

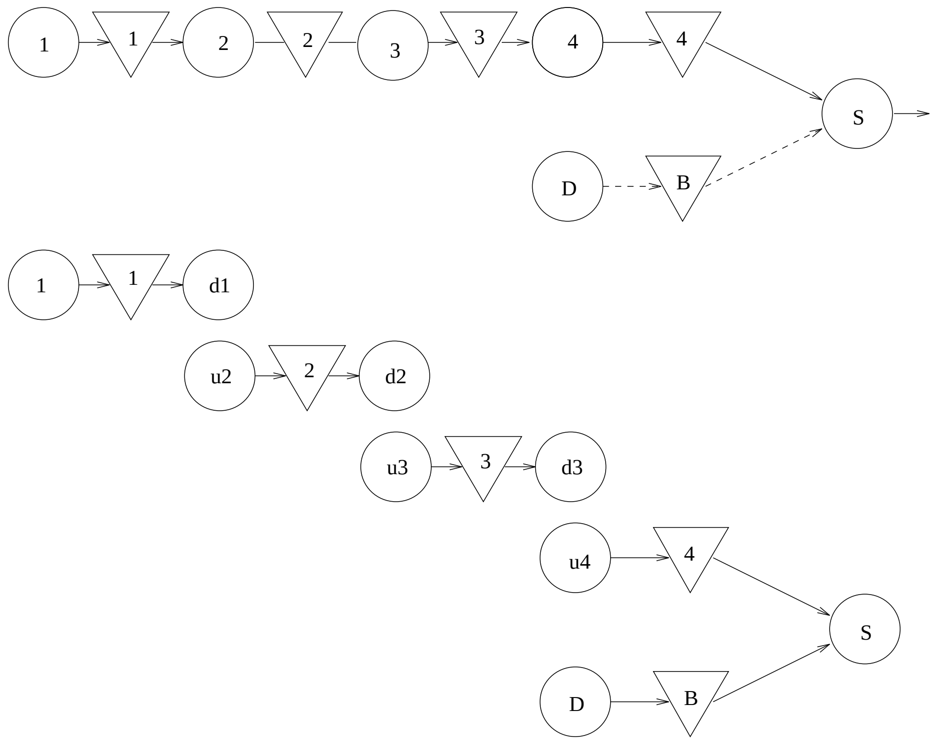

Figure 6-2: Decomposing a kanban line

measures. This report can be as simple as the estimated average production rate and buffer levels; or it can report technical information like the blocking and starvation probabilities and the number of iterations used; or, with some extra input parameters, it can report financial results based on the estimates obtained by the decomposition.

A few implementation tricks are necessary for good performance of this algorithm. During the first few iterations, when conservation of flow is strongly violated, the decomposition equations sometimes produce negative transition rates. It is necessary to test for this and set the rates to reasonable values. After a few iterations, this problem disappears. This remark is also true for the other decompositions in this chapter.

6.2 Kanban model

We now consider the first model that is synchronized to a demand process. For simplicity, we refer to this as a kanban model, but note that kanban control is inherently discrete, and the model deals with a continuous material representation. A more correct name would be a finite buffer, external demand model.

6.2.1 Assembling the kanban model

This line is similar to the tandem line, except that the last stage is synchronized to the demand process. We use the synchronization stage studied in section 4.3 as the last component in the decomposed line, as shown in figure 6-2. The parameters of the two-machine lines and the upstream machine of the synchronization stage are determined as in the tandem case. The exception is the downstream machine of two-machine line $k-1$, the one immediately upstream of the synchronization stage. Its parameters are given by equations (5.37), (5.38), (5.39), and (5.40). These are

$$
\begin{aligned}
\mu_d(k-1) &= \frac{p_{i1}(k)\mu_k + p_{i1}(k)\mu_D}{p_{i1}(k)+p_{i2}(k)} \\
I_d(k-1) &= \frac{\mu_d(k-1)}{\mu_k e_k} + \frac{\mu_d(k-1)}{P(k)} - \frac{\mu_d(k-1)}{\mu_u(k)}(1+I_u(k)) - 1 \\
r_d(k-1) &= \frac{\mathbf{p}(N,1,0,i)\mu_d(k-1)}{P(k)I_d(k-1)} r_D \\
&\quad + \left(1 - \frac{\mathbf{p}(N,1,0,i)\mu_d(k-1)}{P(k)I_d(k-1)}\right) r_k \\
p_d(k-1) &= I_d(k-1)r_d(k-1)
\end{aligned}
$$

We refer to the size of the finished goods buffer as N_k, and the size of the backlog buffer as N_B. The downstream machine of the synchronization stage is the demand process, so its parameters are

$$
\begin{aligned}
\mu_d(k) &= \mu_D \\
r_d(k) &= r_D \\
p_d(k) &= p_D
\end{aligned}
$$

The throughput of the kanban model in saturating demand will be the same as that of the tandem line consisting of machines 1, 2, ..., k. Like in a tandem line, the first machine in the kanban line is never starved. When the demand rate exceeds line capacity, the backlog level goes to infinity. This causes any piece that enters the finished goods buffer to be immediately joined with a waiting demand and leave the system. Therefore, machine k will never be blocked, and the system behaves like the tandem line.

6.2.2 Solving the kanban model

In principle, we use the same algorithm as in the tandem case of section 6.1.2. The only differences are the presence of the finished goods buffer and demand process, and that the last building block in the decomposition is the synchronization stage k. The functions UpdateVariable, PropagateToDownstream, and PropagateToUpstream remain as in section 6.1.2, but procedures TandemReadInput, TandemInitialize, TandemSweepDownstream, and TandemSweepUpstream must be revised. These are the new versions:

procedure KanbanReadInput
begin
    Read k;
    $i := 1$;
    while ($i \leq k$)

begin
    Read μ_i, r_i, p_i, and N_i;
end

Read μ_D, r_D, p_D, and N_B;
end

procedure KanbanInitialize
begin
$i := 1;$
while ($i \leq k - 1$)
begin
$\mu_u(i) := \mu_i;$
$r_u(i) := r_i;$
$p_u(i) := p_i;$
$I_u(i) := p_i / r_i;$

$\mu_d(i) := \mu_{i+1};$
$r_d(i) := r_{i+1};$
$p_d(i) := p_{i+1};$
$I_d(i) := p_{i+1} / r_{i+1};$

$i := i + 1;$
end

$\mu_u(k) := \mu_k;$
$r_u(k) := r_k;$
$p_u(k) := p_k;$
$I_u(k) := p_k / r_k;$

$\mu_d(k) := \mu_D;$
$r_d(k) := r_D;$
$p_d(k) := p_D;$
$I_d(k) := p_D / r_D;$
end

procedure KanbanSweepDownstream
begin
$i := 1;$
while ($i \leq k - 1$)
begin
PropagateToDownstream(i);
$i := i + 1;$
end
end

procedure KanbanSweepUpstream
begin
    EvaluateSynchronizationStage(k);
    UpdateVariable($\mu_d(k-1)$, equation (5.37));
    Calculate $I_d(k-1)$ from equation (5.38);
    UpdateVariable($r_d(k-1)$, equation (5.39));
    UpdateVariable($p_d(k-1)$, equation (5.40));

    $i := k-1$;
    while ($i \geq 2$)
    begin
        PropagateToUpstream(i);
        $i := i-1$;
    end
end

We have here assumed the existence of a function EvaluateSynchronizationStage that updates the performance measures of the synchronization stage based on the current parameter estimates of this stage, as described in section 4.3. We also added code in KanbanSweepUpstream to propagate the behavior of the synchronization stage to the last two-machine line. Otherwise, we have just adjusted the range of the iterations in the code. We can now state the complete algorithm:

procedure KanbanDecomposition
begin
    KanbanReadInput;
    KanbanInitialize;

    MaxDiff := 1;
    while (*MaxDiff* > *Epsilon*)
    begin
        MaxDiff := 0;
        KanbanSweepDownstream;
        KanbanSweepUpstream;
    end

    PrintOutput;
end

6.3 Closed loop

6.3.1 Assembling the closed loop model

A closed loop system is one where a fixed population circulates forever. This could either model a production system where a number of pallets is available for carrying parts, limiting the number of parts in the system at one time to the number of pallets; or it could

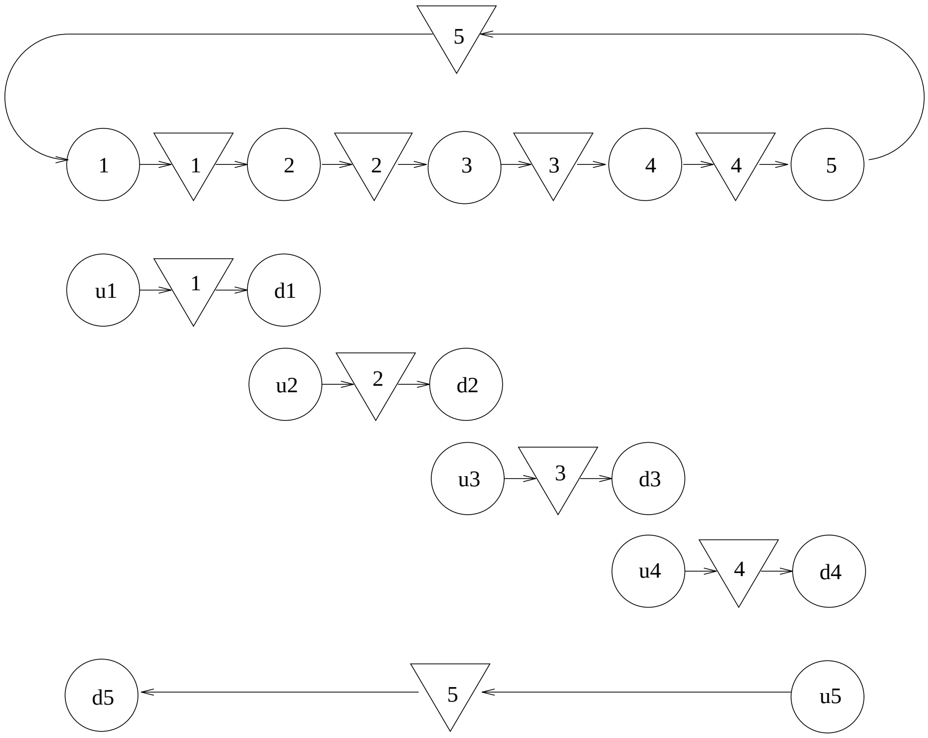

Figure 6-3: Decomposing a closed loop system

model a CONWIP controlled tandem line, where the job release cards accompany parts in the same way as the pallets do.

We decompose this system in the same way as the tandem line, by a series of two-machine lines, related by the equations of section 5.1 (which were repeated in section 6.1.1). But now these lines extend all around the loop, so the two-machine line upstream of the first two-machine line is the last one, as shown in figure 6-3. This leads to a problem: One of the equations used in deriving the decomposition equations is conservation of flow, which reads

$$P(i-1) = P(i) = P(i+1) \text{ for all } i$$

We then freely exchanged the production rates of different stages in the derivation. In the closed loop case, this introduces a dependency in the equations. We need one more equation to get a complete representation.

Such an equation can be found in the fixed loop population. In the decomposition, we cannot guarantee that the loop population constraint is satisfied at all times, since the expressions for buffer level distributions do not consider the levels of any other buffers. But, following the method of Frein, Commault, and Dallery (1994), we can require that the sum of average buffer levels equals the loop population. If the loop population is N_L, this is

$$Q = \sum_{i=0}^{k} \bar{x}_i = N_L \tag{6.2}$$

This equation may introduce a significant error, since a strong constraint on the possible values of the buffer levels at all times is replaced by a much weaker constraint on the average buffer levels. Intuitively, this error is larger for small systems, low values of N_L, or values of N_L that approaches the total buffer space of the system, since the buffer levels are more strongly correlated for these cases. This effect is not accounted for by equation (6.2).

Equation (6.2) is not of the same form as the other decomposition equations, so it must be handled separately. We present an algorithm for this case in section 6.3.2.

We also restate the tandem decomposition equations for the cases where two-machine lines 1 and k interact. We have

$$\mu_u(1) = \frac{p_{o1}(k)\mu_u(k) + p_{o2}(k)\mu_k}{p_{o1}(k) + p_{o2}(k)} \tag{6.3}$$

$$I_u(1) = \frac{\mu_u(1)}{\mu_1 e_1} + \frac{\mu_u(1)}{P(k)} - \frac{\mu_u(1)}{\mu_d(k)}(1 + I_d(k)) - 1 \tag{6.4}$$

$$r_u(1) = \frac{\mathbf{p}(0,0,1,k)\mu_u(k)}{P(1)I_u(1)} r_u(k) + \left(1 - \frac{\mathbf{p}(0,0,1,k)\mu_u(1)}{P(1)I_u(1)}\right) r_1 \tag{6.5}$$

$$p_u(1) = I_u(1) r_u(1) \tag{6.6}$$

$$\mu_d(k) = \frac{p_{i1}(1)\mu_1 + p_{i2}(1)\mu_d(1)}{p_{i1}(1) + p_{i2}(1)} \tag{6.7}$$

$$I_d(k) = \frac{\mu_d(k)}{\mu_1 e_1} + \frac{\mu_d(k)}{P(1)} - \frac{\mu_d(k)}{\mu_u(1)}(1 + I_u(1)) - 1 \tag{6.8}$$

$$r_d(k) = \frac{\mathbf{p}(N,1,0,1)\mu_d(k)}{P(1)I_d(k)} r_d(1) + \left(1 - \frac{\mathbf{p}(N,1,0,1)\mu_d(k)}{P(1)I_d(k)}\right) r_1 \tag{6.9}$$

$$p_d(k) = I_d(k) r_d(k) \tag{6.10}$$

The throughput of the closed loop system is less than that of the corresponding tandem line consisting of machines 1, 2, ..., k. Buffer k will sometimes be empty, so machine 1 will sometimes be starved. If buffer k is also finite, and smaller than the loop population N_L, it will sometimes be full, and machine k will sometimes be blocked. This reduces the throughput compared to a tandem line.

6.3.2 Solving the closed loop model

This case is different from the tandem decomposition in two ways: We have to handle equation 6.2 that is different from the other equations in the system. And there is a dependency in the conservation of flow equations. Following the example of Frein, Commault, and Dallery (1994), we use an algorithm consisting of two nested loops, where the inner loop is similar to the decomposition of a tandem system.

The outer loop is a search procedure to satisfy the loop population constraint. Since we have a dependency, we can discard any one equation from the system that remains when the loop population is fixed. We choose to discard the equation for $I_u(j)$ for some j. Instead, we set a value for $I_u(j)$ in the outer loop, and perform the inner loop decomposition until it converges.

The differences from a tandem decomposition are that $I_u(j)$ is kept at its externally imposed value, and that the upstream machine of the first two-machine line is updated whenever the last two-machine line has been evaluated and *vice versa*. After convergence, we compute the value of the estimated loop population Q, which is the sum of all estimated average buffer levels. This is then compared to the specified loop invariant N_L. If the population is too low, $I_u(j)$ is decreased, and if it is too high, $I_u(j)$ is increased before the next iteration of the outer loop. This procedure is based on the empirical observation that the estimated loop population is monotonically decreasing in $I_u(j)$ for all choices of $j \in \{1, 2, \ldots, k\}$. The algorithm is terminated when the estimated loop population Q is sufficiently close to the loop invariant N_L.

We have implemented the search for $I_u(j)$ as a binary search, where we first find an upper and lower bound on $I_u(j)$. The lower bound is found from the input parameters as $I_u^{\min}(j) = p_j/r_j$. This is because the rest of the system will cause additional idle time at this two-machine line, so $I_u(j)$ cannot be smaller than this value. We find the upper bound by iteratively doubling the value of $I_u(j)$ until the estimated Q is smaller than N_L. The value where this occurs is our upper bound $I_u^{\max}(j)$. The search range is then halved in each subsequent step of the search.

The key to this algorithm is the behavior of the procedure $Q(I_u(j))$. Even if it is monotonically decreasing, it may not be well behaved, which can cause various problems for the search algorithm. We study this further in the next chapter.

We now state the algorithm formally. The procedures UpdateVariable and Propagate-ToUpstream unchanged from section 6.1.2. The simplest modifications are to functions LoopReadInput and LoopInitialize, where only the indexing need to be changed and statements to read the loop population N_L and the search variable index j are added:

procedure LoopReadInput
begin
    Read N_L, j, and k;
    $i := 1$;
    while ($i \leq k$)
    begin
        Read μ_i, r_i, p_i, and N_i;
    end
end

procedure LoopInitialize
begin
    $i := 1$;
    while ($i \leq k$)
    begin
        $\mu_u(i) := \mu_i$;
        $r_u(i) := r_i$;
        $p_u(i) := p_i$;
        if ($i \neq j$)
            $I_u(i) := p_i/r_i$;
        endif
        $i := i + 1$;

```
        end

        i := 1;
        while ( i ≤ k − 1 )
        begin
                μ_d(i) := μ_{i+1};
                r_d(i) := r_{i+1};
                p_d(i) := p_{i+1};
                I_d(i) := p_{i+1}/r_{i+1};

                i := i + 1;
        end

        μ_d(k) := μ_1;
        r_d(k) := r_1;
        p_d(k) := p_1;
        I_d(k) := p_1/r_1;
end
```

The function LoopPropagateToDownstream must test for j before updating $I_u(i)$, since this value is fixed in the outer search loop:

```
procedure LoopPropagateToDownstream(integer i: by value)
begin
        EvaluateTwoMachineLine( i );
        UpdateVariable( μ_u(i + 1), equation (5.7) );
        if ( i + 1 ≠ j )
                Calculate I_u(i + 1) from equation (5.17);
        endif
        UpdateVariable( r_u(i + 1), equation (5.23) );
        UpdateVariable( p_u(i + 1), equation (5.24) );
end
```

The functions LoopSweepDownstream and LoopSweepUpstream are modified to propagate the behavior of two-machine line k to two-machine line 1, and *vice versa*:

```
procedure LoopSweepDownstream
begin
        i := 1;
        while ( i ≤ k − 1 )
        begin
                LoopPropagateToDownstream( i );
                i := i + 1;
        end

        EvaluateTwoMachineLine( k );
        UpdateVariable( μ_u(1), equation (6.3) );
```

```
        Calculate I_u(1) from equation (6.4);
        UpdateVariable( r_u(1), equation (6.5) );
        UpdateVariable( p_u(1), equation (6.6) );
end

procedure LoopSweepUpstream
begin
        i := k;
        while ( i ≥ 2 )
        begin
                PropagateToUpstream( i );
                i := i − 1;
        end

        EvaluateTwoMachineLine( 1 );
        UpdateVariable( μ_d(k), equation (6.7) );
        Calculate I_d(k) from equation (6.8);
        UpdateVariable( r_d(k), equation (6.9) );
        UpdateVariable( p_d(k), equation (6.10) );
end
```

We can now state the fixed point iteration that constitutes the inner loop of the algorithm:

```
procedure LoopInner
begin
        MaxDiff := 1;
        while ( MaxDiff > InnerEpsilon )
        begin
                MaxDiff := 0;
                LoopSweepDownstream;
                LoopSweepUpstream;
        end

        Calculate the loop population estimate Q from equation (6.2);
end
```

We distinguish between the convergence criteria of the inner and outer loops of this algorithm. The *InnerEpsilon* corresponds to the convergence criterion of the tandem decomposition.

Before we can start the binary search for $I_u(j)$, we must find initial bounds $I_u^{\min}(j)$ and $I_u^{\max}(j)$. We express this as two procedures:

```
procedure FindInitialIuMin
begin
        IuMin := p_u(j)/r_u(j);
end
```

procedure FindInitialIuMax
begin
    $I_u(j) := p_u(j)/r_u(j);$
    $Q := 2 * N_L;$
    while ($Q > N_L$)
    begin
        $I_u(j) := 2 * I_u(j);$
        LoopInner;
    end

    $IuMax := I_u(j);$
end

We can now state the complete closed loop decomposition procedure:

procedure LoopDecomposition
begin
    LoopReadInput;
    LoopInitialize;
    FindInitialIuMin;
    FindInitialIuMax;

    while ((*IuMax* - *IuMin*) > *OuterEpsilon*)
    begin
        $I_u(j) := 0.5 * (IuMin + IuMax);$
        LoopInner;
        if ($Q > N_L$)
            $IuMin := I_u(j);$
        else
            $IuMax := I_u(j);$
        endif
    end
end

The variable *OuterEpsilon* is the convergence criterion of the outer loop. This should be larger than the *InnerEpsilon* by one order of magnitude or more to ensure good convergence properties.

Some variations of this algorithm are possible: Frein, Commault, and Dallery (1994) fixed $j = 1$ for all cases. We will discuss the consequences of different choices of j in the next chapter. In the algorithm as we have stated it, we have quietly restarted the next outer iteration from the values of the unknown variables determined by the previous iteration. An alternative is to reinitialize all variables to input parameter values before each call to InnerLoop. The algorithm seems to be faster as stated, but may have slightly better convergence properties if the unknowns are reinitialized between iterations.

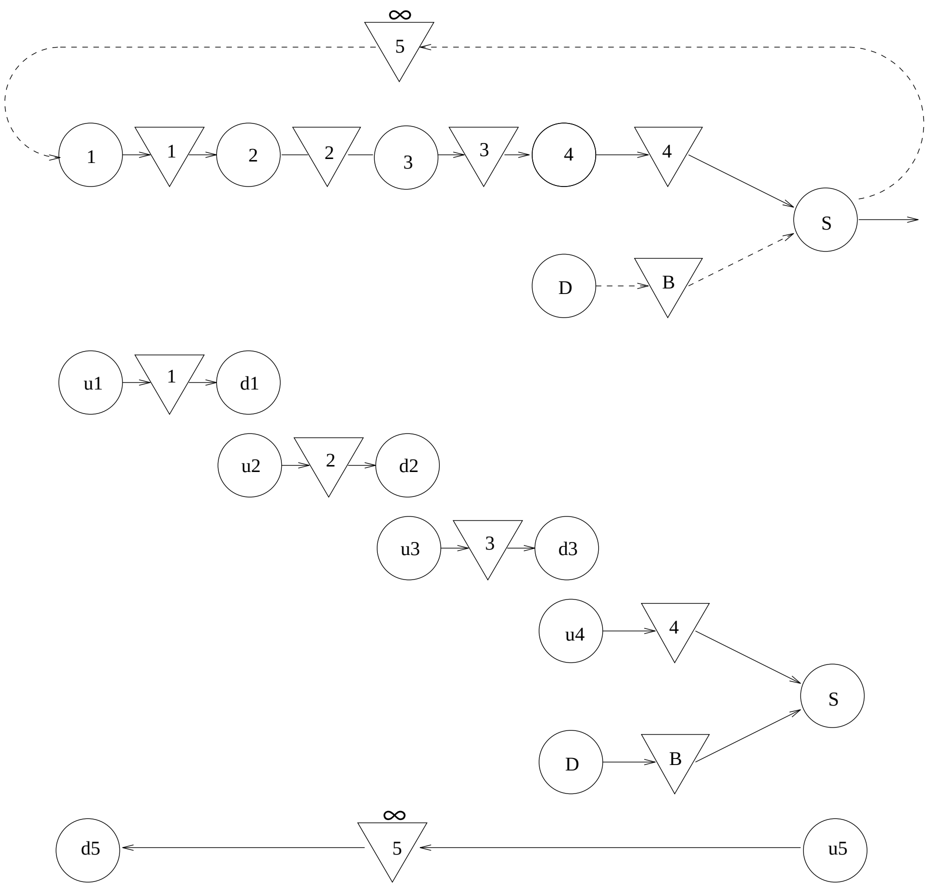

Figure 6-4: Decomposing a hybrid CONWIP/kanban system

6.4 CONWIP/kanban hybrid

6.4.1 Assembling the CONWIP/kanban hybrid model

The CONWIP/kanban hybrid control policy of section 2.6 can also be analyzed by decomposition. It is essentially a combination of the kanban and closed loop representations described in sections 6.2 and 6.3. The CONWIP loop connects the release of jobs to the first machine to completed operations at the synchronization stage, as shown in figure 6-4.

As in the kanban case of section 6.2.1, equations (5.37), (5.39), and (5.40) describe the influence of the synchronization stage on the upstream line. We use the tandem equations (5.7), (5.17), (5.23), (5.24), (5.8), (5.18), (5.25), and (5.26) to describe the tandem part of the line. These equations were restated in section 6.1.1. Equations (5.7), (5.17), (5.23), and (5.24) also determine the parameters of the upstream machine in the synchronization stage.

The parameters of the upstream machine in two-machine line $k+1$, which feeds information back to the first machine in the line, are given by the new synchronization stage decomposition equations (5.33), (5.34), (5.35), and (5.36). For ease of reference, these equations are

$$\begin{aligned}
\mu_u(k+1) &= \frac{p_{o1}(k)\mu_u(k) + p_{o2}(k)\mu_D}{p_{o1}(k) + p_{o2}(k)} \\
I_u(k+1) &= \frac{\mu_u(k+1)}{P(k)} - 1 \\
r_u(k+1) &= \frac{p'_{s1}(k)}{p'_s(k)} r_u(k) + \left(1 - \frac{p'_{s1}(k)}{p'_s(k)}\right) r_D \\
p_u(k+1) &= I_u(k+1) r_u(k+1)
\end{aligned}$$

We restate the tandem decomposition equations for the cases where two-machine lines 1 and $k+1$ interact. We have

$$\begin{aligned}
\mu_u(1) &= \frac{p_{o1}(k+1)\mu_u(k+1) + p_{o2}(k+1)\mu_1}{p_{o1}(k+1) + p_{o2}(k+1)} && (6.11) \\
I_u(1) &= \frac{\mu_u(1)}{\mu_1 e_1} + \frac{\mu_u(1)}{P(k+1)} - \frac{\mu_u(1)}{\mu_d(k+1)}(1 + I_d(k+1)) - 1 && (6.12) \\
r_u(1) &= \frac{\mathbf{p}(0,0,1,k+1)\mu_u(k+1)}{P(1) I_u(1)} r_u(k+1) \\
&\quad + \left(1 - \frac{\mathbf{p}(0,0,1,k+1)\mu_u(1)}{P(1) I_u(1)}\right) r_1 && (6.13) \\
p_u(1) &= I_u(1) r_u(1) && (6.14) \\
\mu_d(k+1) &= \frac{p_{i1}(1)\mu_1 + p_{i2}(1)\mu_d(1)}{p_{i1}(1) + p_{i2}(1)} && (6.15) \\
I_d(k+1) &= \frac{\mu_d(k+1)}{\mu_1 e_1} + \frac{\mu_d(k+1)}{P(1)} - \frac{\mu_d(k+1)}{\mu_u(1)}(1 + I_u(1)) - 1 && (6.16) \\
r_d(k+1) &= \frac{\mathbf{p}(N,1,0,1)\mu_d(k+1)}{P(1) I_d(k+1)} r_d(1) \\
&\quad + \left(1 - \frac{\mathbf{p}(N,1,0,1)\mu_d(k+1)}{P(1) I_d(k+1)}\right) r_1 && (6.17) \\
p_d(k+1) &= I_d(k+1) r_d(k+1) && (6.18)
\end{aligned}$$

As explained in section 5.2, the buffer in this feedback loop must be infinite to prevent interruptions from propagating into the synchronization stage from downstream. Again, the cyclical nature of the system introduces a loop dependency, and we must use the loop population constraint to get an extra equation. For the hybrid CONWIP/kanban policy, we add up the average levels of the internal buffers, the finished goods buffer, and the buffer leading release authorizations back to machine 1. For this case, we let N_C be the loop population as stated in the input file, and Q_C the estimated loop population that results from some choice of $I_u(j)$. The additional equation is then

$$Q_C = \sum_{i=1}^{k+1} \bar{x}_i = N_C \tag{6.19}$$

The throughput of the CONWIP/kanban model in saturating demand is that of the equivalent closed loop system consisting of machines 1, 2, ..., k, with an infinite buffer connecting machines k and 1. As in the kanban line, an infeasible demand rate will cause an infinite backlog, and any piece that enters the finished goods buffer will leave the system immediately, and a release authorization will be passed to buffer $k+1$. Since this buffer is infinite, machine k will never be blocked in saturating demand. As long as the population limit N_C is smaller than the sum of buffer sizes N_1 through N_{k-1}, buffer $k+1$ will sometimes be empty (for instance if machine k is down for a very long time), machine 1 will sometimes be starved, and the throughput will be smaller than that of the corresponding tandem line. If the population limit is greater than this, buffer $k+1$ will always contain release authorizations and machine 1 will never be starved. The throughput in saturating demand then equals that of the tandem line.

6.4.2 Solving the CONWIP/kanban model

This case is different from the closed loop system of section 6.3.2 by having a synchronization stage. We solve the equations through the same approach as for a closed loop system. The procedures UpdateVariable and PropagateToUpstream are unchanged from the tandem case of section 6.1.2. The procedure LoopPropagateToDownstream can be reused from section 6.3.2. The rest must be modified for this case. Again, the simplest modifications are those to CKReadInput and CKInitialize, where we add code to read the parameters of the demand process and to initialize the synchronization stage:

procedure CKReadInput
begin
    Read N_C, j, and k;
    $i := 1$;
    while ($i \leq k$)
    begin
        Read μ_i, r_i, p_i, and N_i;
    end

    Read μ_D, r_D, p_D, and N_B;
end

procedure CKInitialize
begin
    $i := 1$;
    while ($i \leq k$)
    begin
        $\mu_u(i) := \mu_i$;
        $r_u(i) := r_i$;
        $p_u(i) := p_i$;
        if ($i \neq j$)
            $I_u(i) := p_i/r_i$;
        endif

$i := i + 1;$
end

$i := 1;$
while ($i \leq k - 1$)
begin
$\mu_d(i) := \mu_{i+1};$
$r_d(i) := r_{i+1};$
$p_d(i) := p_{i+1};$
$I_d(i) := p_{i+1}/r_{i+1};$

$i := i + 1;$
end

$\mu_u(k) := \mu_k;$
$r_u(k) := r_k;$
$p_u(k) := p_k;$
$I_u(k) := p_k/r_k;$

$\mu_d(k) := \mu_D;$
$r_d(k) := r_D;$
$p_d(k) := p_D;$
$I_d(k) := p_D/r_D;$

$\mu_u(k+1) := \mu_D;$
$r_u(k+1) := r_D;$
$p_u(k+1) := p_D;$
$I_u(k+1) := p_D/r_D;$

$\mu_d(k+1) := \mu_1;$
$r_d(k+1) := r_1;$
$p_d(k+1) := p_1;$
$I_d(k+1) := p_1/r_1;$
end

The functions CKSweepDownstream and CKSweepUpstream are modified from the corresponding loop functions to handle the synchronization stage:

procedure CKSweepDownstream
begin
$i := 1;$
while ($i \leq k - 1$)
begin
LoopPropagateToDownstream(i);
$i := i + 1;$
end

EvaluateSynchronizationStage(k);
UpdateVariable($\mu_u(k+1)$, equation (5.33));
Calculate $I_u(k+1)$ from equation (5.34);
UpdateVariable($r_u(k+1)$, equation (5.35));
UpdateVariable($p_u(k+1)$, equation (5.35));

EvaluateTwoMachineLine($k+1$);
UpdateVariable($\mu_u(1)$, equation (6.11));
Calculate $I_u(1)$ from equation (6.12);
UpdateVariable($r_u(1)$, equation (6.13));
UpdateVariable($p_u(1)$, equation (6.14));
end

procedure CKSweepUpstream
begin
EvaluateSynchronizationStage(k);
UpdateVariable($\mu_d(k-1)$, equation (5.37));
Calculate $I_d(k-1)$ from equation (5.38);
UpdateVariable($r_d(k-1)$, equation (5.39));
UpdateVariable($p_d(k-1)$, equation (5.39));

$i := k-1$;
while ($i \geq 2$)
begin
PropagateToUpstream(i);
$i := i-1$;
end

EvaluateTwoMachineLine(1);
UpdateVariable($\mu_d(k+1)$, equation (6.15));
Calculate $I_d(k+1)$ from equation (6.16);
UpdateVariable($r_d(k+1)$, equation (6.17));
UpdateVariable($p_d(k+1)$, equation (6.18));
end

The inner fixed point iteration of the algorithm is now a very simple modification of LoopInner: We just call the redefined procedures CKSweepDownstream and CKSweepUpstream instead of the loop versions:

procedure CKInner
begin
$MaxDiff := 1$;
while ($MaxDiff > InnerEpsilon$)
begin
$MaxDiff := 0$;

```
        CKSweepDownstream;
        CKSweepUpstream;
    end

    Calculate the loop population estimate Q_C from equation (6.19);
end
```

The procedure FindInitialIuMin is unchanged from the closed loop case, but we must update the search for the initial $I_u^{\max}(j)$:

```
procedure CKFindInitialIuMax
begin
    I_u(j) := p_u(j)/r_u(j);
    Q_C := 2 * N_C;
    while ( Q_C > N_C )
    begin
        I_u(j) := 2 * I_u(j);
        CKInner;
    end

    IuMax := I_u(j);
end
```

We can now state the complete hybrid CONWIP/kanban decomposition procedure:

```
procedure CKDecomposition
begin
    CKReadInput;
    CKInitialize;
    FindInitialIuMin;
    CKFindInitialIuMax;

    while ( (IuMax - IuMin) > OuterEpsilon )
    begin
        I_u(j) := 0.5 * ( IuMin + IuMax );
        CKInner;
        if ( Q_C > N_C )
            IuMin := I_u(j);
        else
            IuMax := I_u(j);
        endif
    end
end
```

As for the loop case of section 6.3.2, we can alternatively reinitialize all unknowns to input parameter values between iterations. This can be done by calling CKInitialize as the first statement in CKInner, and will trade some speed for better convergence properties.

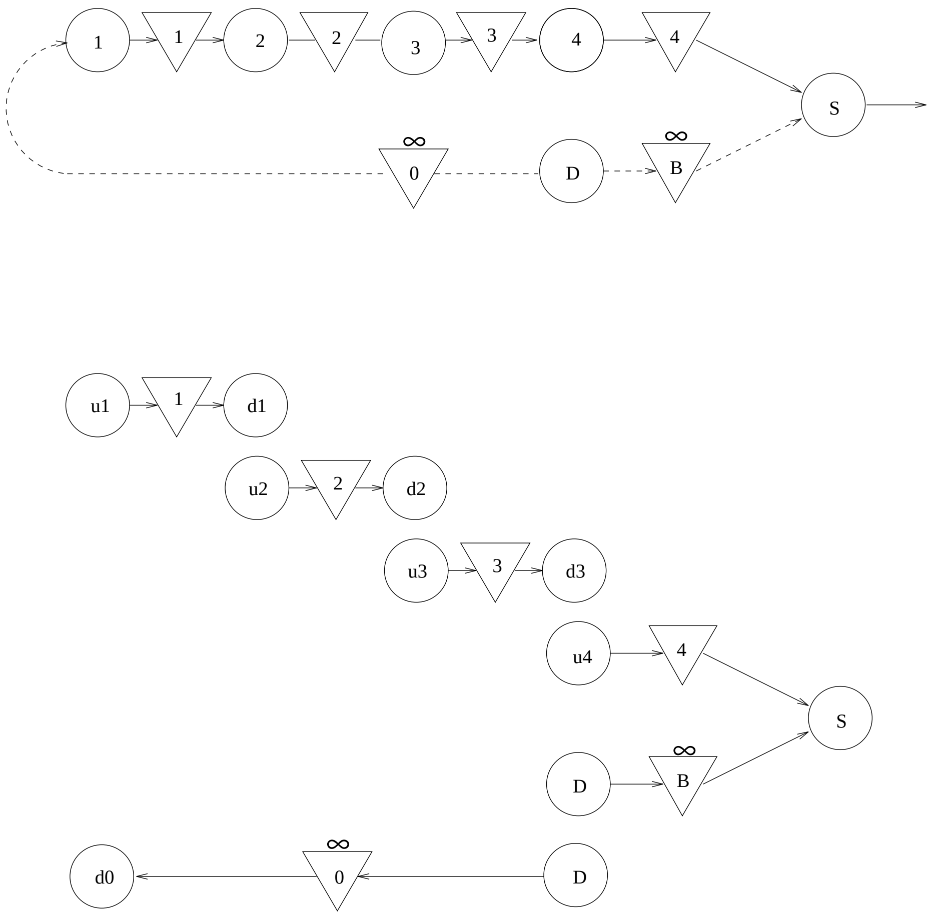

Figure 6-5: Decomposing a hybrid basestock/kanban system

6.5 Basestock/kanban hybrid

6.5.1 Assembling the basestock/kanban hybrid model

We restrict ourselves to basestock policies where the basestock levels of all internal buffers are zero, or, equivalently, policies where all hedging points are equal. This limits the number of loops in the system to one, which simplifies the solution procedure. We also restrict this case to systems with unlimited backlogging.

The basestock control differs from CONWIP in the timing of job releases to the first machine. In a basestock policy, new jobs are released to the first machine when demands arrive; while in CONWIP, a new job is released when finished goods depart from the system. This is illustrated in the upper part of figure 6-5, where an information loop is shown going from the demand process to the first machine. This system can be decomposed to a collection of the building blocks we have developed, as shown in the figure.

There are two points to note about this figure: The flow of information to machine 1 does not come from the departures of material from the synchronization stage as in CONWIP, but comes directly from the demand process. This means that we do *not* use equations

(5.33), (5.34), (5.35), and (5.36), since these describe the upstream machine of two-machine line $k+1$ in terms of the output process from the synchronization stage. Instead, we have a two-machine line 0 with an infinite buffer and the demand process as its upstream machine. Then we have that

$$\begin{aligned} \mu_u(0) &= \mu_d(k) = \mu_D \\ I_u(0) &= I_d(k) = p_D/r_D \\ r_u(0) &= r_d(k) = r_D \\ p_u(0) &= p_d(k) = p_D \end{aligned}$$

This duplication of the demand process into two independent "machines" will introduce some approximation error, especially for small systems. In the original system, there is only one demand process, and if it is stopped, the flow into both machine 1 and the synchronization stage stops. The decomposition does not capture this correlation.

As in the kanban case of section 6.2.1, we use equations (5.37), (5.39), and (5.40) to describe the influence of the synchronization stage on the upstream line. The rest of the system is described by the tandem equations (5.7), (5.23), (5.24), (5.8), (5.25), and (5.26), which were restated in section 6.1.1.

Again, we have a loop introducing a dependency in the decomposition equations. The loop in this control policy is different from the CONWIP loop. Going clockwise around the loop, starting from the demand process, one notices some arrows pointing in the counter-clockwise direction. These represent the flow of information from the demand process to the synchronization "machine." The loop invariant is not a constant number of jobs in the loop, but a constant difference between the number of jobs heading clockwise and counter-clockwise in the loop. That is, when the finished goods buffer is filled up, its level equals its basestock level, and the rest of the system is empty. When a demand arrives, it is, at least conceptually, translated into a *release authorization* and a *withdrawal request*. The release authorization is sent to the first machine to allow it to introduce one unit of material to the line. The withdrawal request is sent to the synchronization stage to be joined with a unit of finished goods. This maintains the difference between the two directions. When a part is delivered from the system, one unit of finished goods is joined with one withdrawal request, and both depart from the system. This, too, maintains the loop invariant.

As in the closed loop system (section 6.3) and the CONWIP/kanban hybrid system (section 6.4), we cannot handle this loop invariant directly. Instead, we require that the average buffer levels obey the loop invariant. We then have the additional equation

$$Q_B = \sum_{i=0}^{k} \bar{x}_i - \bar{x}_B = Z_B \tag{6.20}$$

where $\bar{x}_B$ is the average system backlog and Z_B is the basestock level of the system. Since both buffer 0 and the backlog buffer are infinite, Q_B can take on values from minus infinity to plus infinity.

As in the previous cases of sections 6.3 and 6.4, this introduces an approximation error due to the replacement of the loop invariant with a less restrictive constraint on the average buffer levels.

The throughput of the basestock/kanban model in saturating demand equals that of the corresponding tandem line consisting of machines 1 - k. If the demand rate is infeasible, both the backlog and buffer 0 will go to infinity for any choice of the basestock level Z_B. Any piece that enters the finished goods buffer will then immediately be joined with a waiting demand and leave the system, so machine k is never blocked. Whenever machine 1 is ready to do an operation, it has an infinite number of release authorizations, so it is never starved. The system then behaves like the tandem line.

6.5.2 Solving the basestock/kanban hybrid model

Since the demand process is duplicated on the input end of the decomposed system, the resulting collection of two-machine lines and a synchronization stage is the same as a kanban line with an infinite buffer upstream and a population constraint imposed on it. The procedures to read input and initialize the two-machine lines are straightforward:

procedure BKReadInput
begin
    Read Z_B, j, and k;
    $i := 1$;
    while ($i \leq k$)
    begin
        Read μ_i, r_i, p_i, and N_i;
    end

    Read μ_D, r_D, p_D, and N_B;
end

procedure BKInitialize
begin
    $i := 1$;
    while ($i \leq k$)
    begin
        $\mu_u(i) := \mu_i$;
        $r_u(i) := r_i$;
        $p_u(i) := p_i$;
        if ($i \neq j$)
            $I_u(i) := p_i/r_i$;
        endif
        $i := i + 1$;
    end

    $i := 1$;
    while ($i \leq k - 1$)
    begin
        $\mu_d(i) := \mu_{i+1}$;
        $r_d(i) := r_{i+1}$;

```
        p_d(i) := p_{i+1};
        I_d(i) := p_{i+1}/r_{i+1};

        i := i + 1;
    end

    μ_u(k) := μ_k;
    r_u(k) := r_k;
    p_u(k) := p_k;
    I_u(k) := p_k/r_k;

    μ_d(k) := μ_D;
    r_d(k) := r_D;
    p_d(k) := p_D;
    I_d(k) := p_D/r_D;

    N(0) := ∞;
    μ_u(0) := μ_D;
    r_u(0) := r_D;
    p_u(0) := p_D;
    I_u(0) := p_D/r_D;

    μ_d(0) := μ_1;
    r_d(0) := r_1;
    p_d(0) := p_1;
    I_d(0) := p_1/r_1;
end
```

The downstream and upstream sweeps are like those of the kanban decomposition, except that they include two-machine line 0, and avoid changing the value of $I_u(j)$:

```
procedure BKSweepDownstream
begin
    i := 0;
    while ( i ≤ k − 1 )
    begin
        LoopPropagateToDownstream( i );
        i := i + 1;
    end
end

procedure BKSweepUpstream
begin
    EvaluateSynchronizationStage( k );
    UpdateVariable( μ_d(k − 1), equation (5.37) );
    Calculate I_d(k − 1) from equation (5.38);
```

UpdateVariable($r_d(k-1)$, equation (5.39));
UpdateVariable($p_d(k-1)$, equation (5.40));

$i := k - 1$;
while ($i \geq 1$)
begin
PropagateToUpstream(i);
$i := i - 1$;
end
end

When these procedures are defined, we can state the inner loop of the algorithm:

procedure BKInner
begin
MaxDiff := 1;
while (*MaxDiff* > *InnerEpsilon*)
begin
MaxDiff := 0;
BKSweepDownstream;
BKSweepUpstream;
end

Calculate the loop population estimate Q_B from equation (6.20);
end

The procedure FindInitialIuMin is unchanged from the closed loop case of section 6.3.2, but we must update the search for the initial $I_u^{\max}(j)$:

procedure BKFindInitialIuMax
begin
$I_u(j) := p_u(j)/r_u(j)$;
$Q_C := 2 * N_C$;
while ($Q_C > N_C$)
begin
$I_u(j) := \mathit{StepSize} * I_u(j)$;
BKInner;
end

$\mathit{IuMax} := I_u(j)$;
end

Note that whe have used a variable *StepSize* to determine how much to increase the value of $I_u(j)$ in each iteration. For the CONWIP/kanban case, the step size was fixed at 2. In this case, the loop population estimate will go to minus infinity when $I_u(j)$ becomes sufficiently large, so the step size should be smaller to get a better initial estimate. We have used step sizes of 1.1 and 1.01 in our numerical experiments.

We can now state the complete hybrid basestock/kanban decomposition procedure:

```
procedure BKDecomposition
begin
    BKReadInput;
    BKInitialize;
    FindInitialIuMin;
    BKFindInitialIuMax;

    while ( (IuMax - IuMin) > OuterEpsilon )
    begin
        I_u(j) := 0.5 * ( IuMin + IuMax );
        BKInner;
        if ( Q_C > N_C )
            IuMin := I_u(j);
        else
            IuMax := I_u(j);
        endif
    end
end
```

As the numerical results in section 7.5 show, this basestock/kanban decomposition does not have very good convergence properties. It is an interesting topic for future research to find a better algorithm for solving the decomposition equations, or to find a more stable set of equations for this particular case.

Chapter 7

Numerical results

We are now approaching the critical question: "Yes, but does it work?" The purpose of this chapter is to present a collection of results that exhibit what we know about the numerical behavior of the decompositions we have presented. We look at convergence, speed, and accuracy. The results are compared to simulations and other decompositions. In section 7.1 we treat tandem systems, in section 7.3 kanban lines, and in section 7.2 closed loop systems. We treat the hybrid CONWIP/kanban decomposition in section 7.4 and the hybrid basestock/kanban decomposition in section 7.5. Finally, we use the different decompositions to compare alternate control policies applied to the same line in section 7.6.

7.1 Tandem lines

A good collection of inhomogeneous tandem line test cases is found in Le Bihan (1994). His F series of experiments compare several analytic approaches for five-machine inhomogeneous lines. The ten experiments in the series differ in the processing rates of different machines in the line. In tables 7.1 through 7.10, we continue these experiments. Each table contains the input values for the case, and the average production rate P and average buffer levels $\bar{n}_i$ estimated by the various methods. In the column marked "evals" we report the number of two-machine evaluations for those decompositions where this was available.

In each table, we have listed Le Bihan's simulation results (key LBsim in the tables), his result for the decomposition method developed by Semery (1987) (key Sem), and his result for the method by Alvarez Vargas, Dallery, and David (1991a) (key ADD).

We have added a new set of continuous material simulations with 95% confidence intervals (key Csim). We ran 15 independent replications of each simulation, where each run had a duration of 10^6 time units and an initial warm-up period of 10^4 time units. The confidence intervals were computed from a Student's t-distribution with 14 degrees of freedom, as

$$x \in \bar{X} \pm t_{14}^{.025} \frac{S^2}{\sqrt{n}} \tag{7.1}$$

where x is some performance measure, $\bar{X}$ the sample mean of that performance measure, S^2 the sample variance, and n the sample size. (See also section 4.2.4.)

The entries marked Dsim come from a discrete material simulation of the same line. The

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.111	15
2	.05	.005	1.667	20
3	.2	.02	1.0	10
4	.1	.01	1.428	15
5	.08	.01	1.250	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.857	12.19	18.67	1.18	2.62	
Csim	0.857 ± 0.001	12.20 ± 0.03	18.69 ± 0.03	1.19 ± 0.01	2.63 ± 0.02	
Dsim	0.853 ± 0.002	12.05 ± 0.06	19.01 ± 0.03	2.02 ± 0.02	3.20 ± 0.03	
Sem	0.863	8.51	18.31	1.21	3.66	
ADD	0.862	13.25	18.95	1.07	2.06	
ADDX	0.862	13.25	18.95	1.07	2.06	54
BDDX	0.862	13.25	18.95	1.07	2.06	48

Table 7.1: Case F1

operation times were deterministic with value $1/\mu_i$ for machine i. The line was operated by a blocking before service discipline, and the buffer levels shown includes the average amount of material in the buffer plus the average amount held in the machine downstream of the buffer. As shown by Berkley (1991), this is equivalent to a line controlled by blocking after service, where all buffer sizes are reduced by one.

Entries marked ADDX were generated by the decomposition procedure developed by Burman (1995), also referred to as *Accelerated DDX*. To make the results comparable, we used the same convergence criterion as in this thesis. The ADDX algorithm, as implemented here, was terminated when the maximal relative change of any unknown variable became less than 10^{-5} beteeen two iterations. This is different from the criterion used by Burman.

Finally, the entries modestly marked BDDX were generated by the tandem decomposition procedure developed in this thesis.

We note that our decomposition procedure converged in every case. It also converged with less computational effort than the accelerated DDX developed by Burman. In case F9, Le Bihan reported that the decomposition by Alvarez Vargas, Dallery, and David failed to converge. All the decompositions tend to overestimate the production rate (except Semery's, where there is no clear pattern), and our procedure sometimes overestimates by more than the other decompositions. When a continuous material decomposition is used to analyze a discrete material production system, this could be a problem, since the continuous material model already gives a higher production rate than the discrete material asynchronous model, as shown by the simulation results in the tables. The average buffer level estimates from the decompositions are less accurate than the production rate estimates, and our decomposition has comparable accuracy to the other decompositions in this respect. We note that the buffer level estimates seem to be more accurate when the estimated level is close to having the buffer full or empty, than when the estimated level is away from these boundaries.

In some other cases, our decomposition procedure does not converge, but cycles forever between two or more solutions, neither of which satisfies conservation of flow. The oscillation takes place near the correct solution; typically, the difference in production rates between

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.250	15
2	.05	.005	1.111	20
3	.2	.02	1.667	10
4	.1	.01	1.0	15
5	.08	.01	1.428	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.858	13.92	15.02	8.84	1.32	
Csim	0.859 ± 0.001	13.93 ± 0.01	15.03 ± 0.03	8.84 ± 0.01	1.32 ± 0.01	
Dsim	0.854 ± 0.002	14.01 ± 0.03	15.13 ± 0.07	9.23 ± 0.02	1.90 ± 0.02	
Sem	0.833	13.96	10.63	8.48	1.25	
ADD	0.868	13.95	15.93	9.12	1.34	
ADDX	0.868	13.95	15.93	9.18	1.34	48
BDDX	0.868	13.95	15.93	9.18	1.34	42

Table 7.2: Case F2

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.428	15
2	.05	.005	1.250	20
3	.2	.02	1.111	10
4	.1	.01	1.667	15
5	.08	.01	1.0	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.858	14.32	17.62	5.35	11.96	
Csim	0.857 ± 0.001	14.32 ± 0.01	17.60 ± 0.03	5.35 ± 0.03	11.97 ± 0.03	
Dsim	0.855 ± 0.001	14.40 ± 0.02	17.77 ± 0.05	5.53 ± 0.04	12.31 ± 0.06	
Sem	0.835	14.30	17.82	3.15	10.78	
ADD	0.873	14.35	17.60	5.55	13.55	
ADDX	0.873	14.35	17.60	5.54	13.55	108
BDDX	0.873	14.35	17.60	5.54	13.55	84

Table 7.3: Case F3

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.0	15
2	.05	.005	1.428	20
3	.2	.02	1.250	10
4	.1	.01	1.111	15
5	.08	.01	1.667	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.894	2.85	8.05	5.23	1.06	
Csim	0.894 ± 0.001	2.86 ± 0.05	8.02 ± 0.10	5.22 ± 0.03	1.07 ± 0.01	
Dsim	0.889 ± 0.002	3.67 ± 0.07	8.77 ± 0.15	5.76 ± 0.06	1.58 ± 0.02	
Sem	0.905	2.51	8.97	5.41	1.06	
ADD	0.912	1.78	6.89	5.00	1.08	
ADDX	0.912	1.71	6.03	5.27	1.08	96
BDDX	0.912	1.64	5.60	5.11	1.08	84

Table 7.4: Case F4

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.053	15
2	.05	.005	1.250	20
3	.2	.02	1.0	10
4	.1	.01	1.176	15
5	.08	.01	1.111	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.875	14.65	4.89	4.37	6.98	
Csim	0.875 ± 0.001	14.65 ± 0.01	4.90 ± 0.05	4.37 ± 0.03	7.00 ± 0.05	
Dsim	0.872 ± 0.002	14.99 ± 0.01	5.70 ± 0.10	5.00 ± 0.04	7.48 ± 0.07	
Sem	0.864	14.66	5.44	5.83	7.56	
ADD	0.892	14.64	3.14	3.46	6.83	
ADDX	0.893	14.64	3.06	2.98	7.17	72
BDDX	0.894	14.64	2.89	2.83	6.95	66

Table 7.5: Case F5

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.053	15
2	.05	.005	1.250	20
3	.2	.02	1.0	10
4	.1	.01	1.176	15
5	.08	.01	1.111	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.838	11.19	17.49	2.12	4.02	
Csim	0.837 ± 0.001	11.20 ± 0.04	17.47 ± 0.05	2.11 ± 0.01	4.02 ± 0.03	
Dsim	0.832 ± 0.002	11.27 ± 0.06	17.68 ± 0.05	2.86 ± 0.02	4.57 ± 0.05	
Sem	0.845	9.30	17.33	2.08	4.46	
ADD	0.845	12.35	17.92	1.93	3.41	
ADDX	0.845	12.36	17.92	1.93	3.41	54
BDDX	0.845	12.36	17.92	1.93	3.41	48

Table 7.6: Case F6

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.111	15
2	.05	.005	1.053	20
3	.2	.02	1.250	10
4	.1	.01	1.0	15
5	.08	.01	1.176	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.842	13.00	13.67	7.93	2.33	
Csim	0.841 ± 0.001	13.00 ± 0.02	13.71 ± 0.07	7.94 ± 0.02	2.45 ± 0.02	
Dsim	0.837 ± 0.002	13.08 ± 0.06	14.06 ± 0.06	8.25 ± 0.03	3.06 ± 0.03	
Sem	0.837	13.13	11.41	7.89	2.35	
ADD	0.850	13.09	14.28	8.05	2.44	
ADDX	0.851	13.07	14.13	8.27	2.45	54
BDDX	0.853	13.04	14.13	8.35	2.47	48

Table 7.7: Case F7

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.176	15
2	.05	.005	1.111	20
3	.2	.02	1.053	10
4	.1	.01	1.250	15
5	.08	.01	1.0	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.837	13.58	16.45	4.20	9.59	
Csim	0.837 ± 0.001	13.57 ± 0.02	16.47 ± 0.05	4.24 ± 0.03	9.63 ± 0.05	
Dsim	0.832 ± 0.002	13.62 ± 0.04	16.65 ± 0.07	4.64 ± 0.05	9.89 ± 0.08	
Sem	0.832	13.65	16.68	3.17	9.47	
ADD	0.848	13.70	16.47	4.10	9.69	
ADDX	0.852	13.68	16.41	3.26	10.63	102
BDDX	0.862	13.62	16.23	3.03	10.90	96

Table 7.8: Case F8

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.0	15
2	.05	.005	1.176	20
3	.2	.02	1.111	10
4	.1	.01	1.053	15
5	.08	.01	1.250	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.861	6.00	12.47	5.57	2.19	
Csim	0.861 ± 0.001	6.03 ± 0.05	12.55 ± 0.09	5.58 ± 0.03	2.20 ± 0.03	
Dsim	0.855 ± 0.002	6.54 ± 0.08	12.81 ± 0.13	5.98 ± 0.04	2.78 ± 0.02	
Sem	0.875	5.61	12.81	5.65	2.17	
ADD						
ADDX	0.878	5.47	13.80	5.90	2.21	102
BDDX	0.882	5.90	14.58	6.05	2.23	96

Table 7.9: Case F9

i	r_i	p_i	μ_i	N_i
1	.2	.0125	1.250	15
2	.05	.005	1.0	20
3	.2	.02	1.176	10
4	.1	.01	1.111	15
5	.08	.01	1.053	

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
LBsim	0.849	14.16	8.98	5.44	7.22	
Csim	0.849 $\pm$ 0.001	14.15 $\pm$ 0.01	9.00 $\pm$ 0.06	5.45 $\pm$ 0.03	7.20 $\pm$ 0.06	
Dsim	0.845 $\pm$ 0.002	14.32 $\pm$ 0.02	9.61 $\pm$ 0.08	5.93 $\pm$ 0.03	7.60 $\pm$ 0.06	
Sem	0.852	14.18	8.71	5.67	6.99	
ADD	0.862	14.15	8.50	4.98	6.99	
ADDX	0.864	14.15	8.00	5.30	7.32	66
BDDX	0.865	14.14	7.98	5.36	7.40	60

Table 7.10: Case F10

i	r_i	p_i	μ_i	N_i
1	0.2	0.0125	1.0	15
2	0.05	0.005	1.176	20
3	0.2	0.02	1.111	10
4	0.1	0.01	0.9	15
5	0.08	0.01	1.25	

Table 7.11: A problematic tandem case

the two-machine lines is less than 5%. This seems to happen when the isolated production rates of adjacent machines in the line become too different. An example of a line that causes cycling is given in table 7.11. Burman's ADDX decomposition seems more robust. When given this input file, it converges in just 54 two-machine evaluations.

7.2 Closed loop

7.2.1 Convergence behavior

To illustrate the behavior of closed loop decompositions, we first present a simple four-machine loop where one machine is different from the others. The system description is in table 7.12. Recall from section 6.3.2 that the solution algorithm for this case uses an outer loop that is a binary search in the value of $I_u(j)$, which is the ratio of failure and repair rates in the upstream machine of two-machine line j in the decomposition. We now solve the system with each choice of $j = 1, 2, 3, 4$.

The results are listed in tables 7.13 through 7.16. We compare the results of a continuous material simulation of the system (key Csim) with loop decompositions based both on the decomposition equations of Burman (1995) (key ADDX), and on the equations developed

i	r_i	p_i	μ_i	N_i
1	.15	.015	0.8	10
2	.1	.01	1.0	10
3	.1	.01	1.0	10
4	.1	.01	1.0	10

Table 7.12: A simple loop, population 20

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
Csim	0.687 ± 0.001	1.48 ± 0.01	2.35 ± 0.01	7.65 ± 0.01	8.52 ± 0.01	
ADDX	0.657	2.17	3.64	6.36	7.83	5298
BDDX	0.661	2.07	3.56	6.44	7.93	5484

Table 7.13: Case L1: $j = 1$

in this thesis (key BDDX).

As expected, they are all equivalent, except that some choices of j require twice as much computational effort as others. In all of these cases, the decompositions underestimate the production rate, and tend to underestimate the differences in average buffer level between adjacent buffers. That is, the decompositions seem to predict a more even distribution of material in the system than what the simulations report. The number of two-machine evaluations required to solve the system is about two orders of magnitude larger than what was necessary for the tandem lines in the previous section.

We now investigate what causes the difference in the number of two-machine evaluations for different choices of j. Recall again that we search in $I_u(j)$ for some j. For each value chosen, we perform the inner loop of the algorithm until convergence. From the resulting solution, we add up all the average buffer levels to get an estimated loop population Q. This will be different for different choices of $I_u(j)$, so we can view Q as a function of $I_u(j)$. This is a different function for each choice of j, so we refer to it as $Q(I_u(j), j)$.

To explore the behavior of the function $Q(I_u(j), j)$ we made a modified version of the loop decomposition program. This version does not perform a search in $I_u(j)$, but moves in small increments between two specified values of $I_u(j)$ for a particular choice of j. For each value visited, it performs the inner loop of the decomposition until convergence. It then prints the values of j, $I_u(j)$, and the resulting value of $Q(I_u(j), j)$, before incrementing

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
Csim	0.687 ± 0.000	1.48 ± 0.01	2.36 ± 0.02	7.65 ± 0.02	8.51 ± 0.01	
ADDX	0.657	2.17	3.64	6.36	7.83	8316
BDDX	0.661	2.07	3.56	6.44	7.93	8604

Table 7.14: Case L2: $j = 2$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_2$	evals
Csim	0.688 ± 0.001	1.48 ± 0.01	2.33 ± 0.01	7.66 ± 0.01	8.53 ± 0.01	
ADDX	0.657	2.17	3.64	6.36	7.83	4956
BDDX	0.661	2.07	3.56	6.44	7.93	4602

Table 7.15: Case L3: $j = 3$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	evals
Csim	0.687 ± 0.001	1.48 ± 0.01	2.36 ± 0.02	7.65 ± 0.02	8.51 ± 0.01	
ADDX	0.657	2.17	3.64	6.36	7.83	3756
BDDX	0.661	2.07	3.56	6.44	7.93	3588

Table 7.16: Case L4: $j = 4$

$I_u(j)$.

We used this programs to plot the function $Q(I_u(j), j)$. Each of the four curves in figure 7-1 shows this function for a particular choice of j. We have also plotted the points visited by the decomposition search along these curves. Superimposed on the plot, there is a horizontal line that shows the loop population constraint $N_L = 20$. We see from the plot how the searches converge on the correct loop population in all four cases, but that the different curves cross the line $Q = 20$ at different slopes.

Note that there are some vertical discontinuities in these curves. The exact locations of the discontinuities depend on the initial conditions of the solution procedure. In these cases, we have reinitialized all unknown variables to input values between each new value of $I_u(1)$. The horizontal section of the curve for $j = 2$, ranging approximately from $I_u(2) = .18$ to .20, covers a section where the inner loop decomposition did not converge, but cycled between several solutions. We do not fully understand the behavior of these functions.

In figure 7-2, we have shown similar curves as generated by Burman's ADDX equations applied to the same closed system. The general shapes of the functions are the same as in figure 7-1, but the curves generated by ADDX are devoid of discontinuities. This could explain the more robust behavior of the ADDX algorithm.

If successive machines in the line are very different, for instance as in the homogeneous system shown in table 7.17, all the solution algorithms run into trouble for some values of the loop population constraint. The problem is that the function $Q(I_u(1), 1)$ has a large discontinuity, as shown in figure 7-3. This figure was generated by the ADDX equations. Near the discontinuity, even a very small step in $I_u(1)$ causes a large change in Q. This causes the search algorithm to fail. It was not possible to investigate the behavior of $Q(I_u(1), 1)$ for $I_u(1) < 0.5$ because of numerical overflow in the two-machine evaluation.

7.2.2 Varying line length

Next we want to look at the impact of varying line length. Since we consider a closed loop, the levels of individual buffers are more strongly correlated than in the tandem line. For

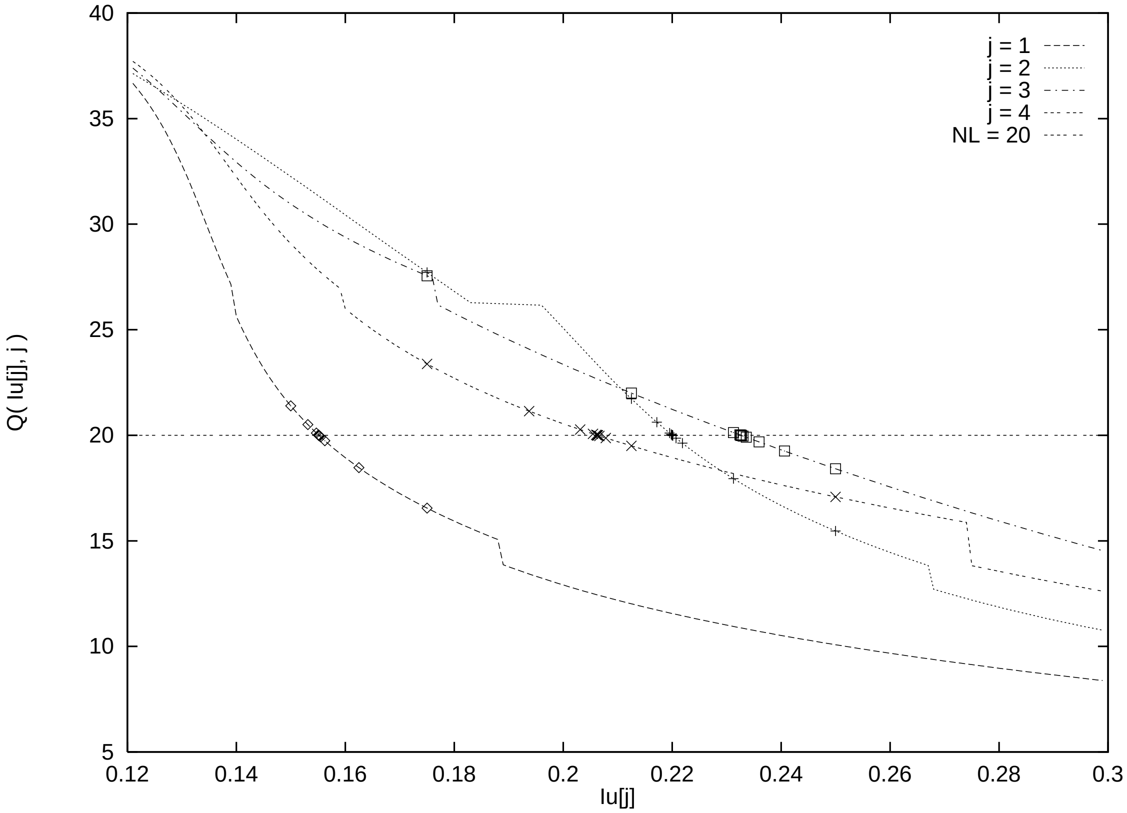

Figure 7-1: The estimated loop population Q as a function of $I_u(1)$

i	r_i	p_i	μ_i	N_i
1	1	.5	1.0	20
2	.1	.01	1.0	5
3	.2	.01	1.0	10
4	.1	.01	1.0	10
5	.2	.01	1.0	10
6	.1	.01	1.0	15

Table 7.17: A problematic loop case

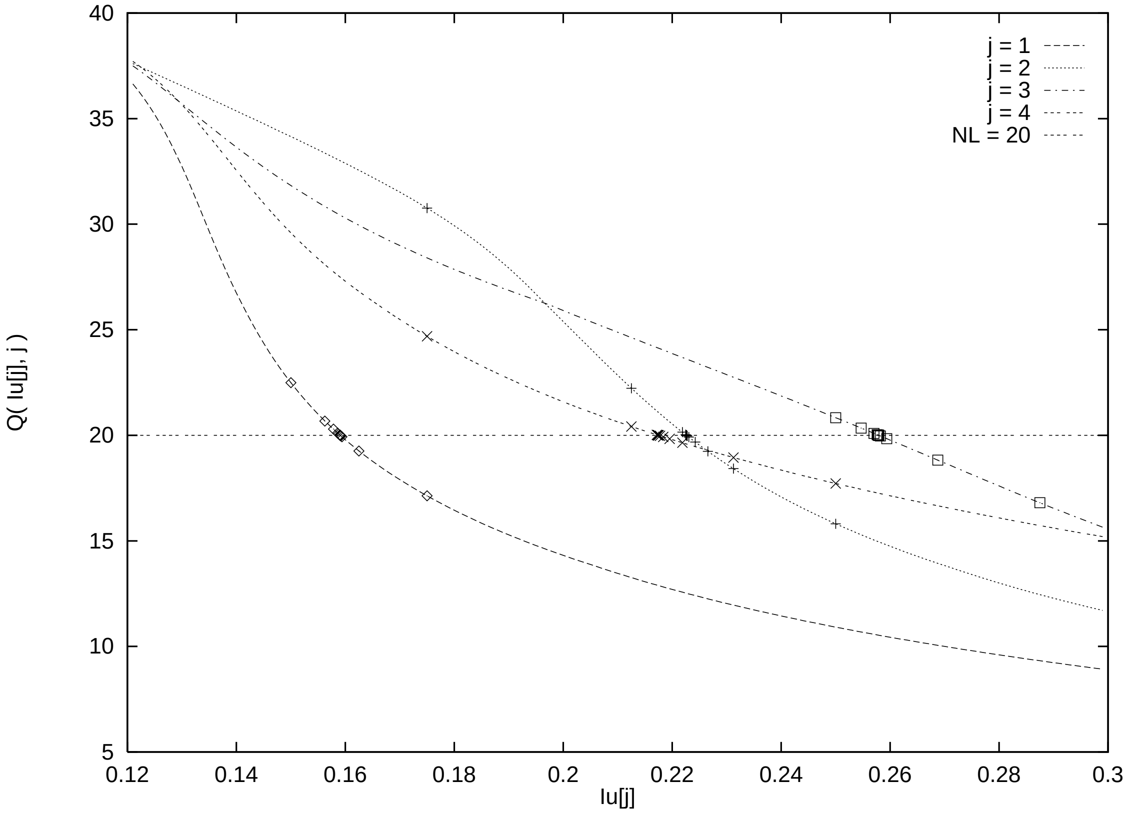

Figure 7-2: The loop population estimated by ADDX as a function of $I_u(1)$

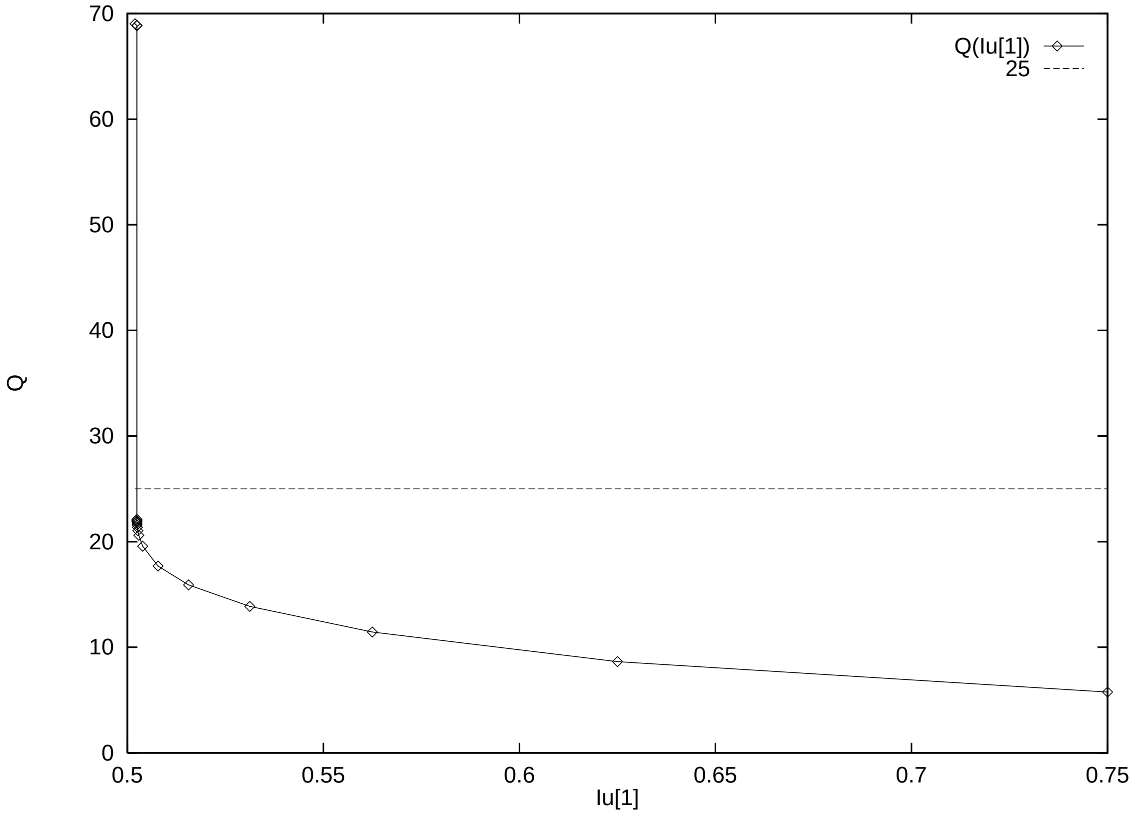

Figure 7-3: ADDX estimates of loop population for the problematic case

instance, if the loop population is N_L, and some buffer has reached this level, we know that all other buffers must be empty. The decompositions do not represent this. As noted by Frein, Commault, and Dallery (1994), this leads to inaccurate solutions when analyzing small systems.

To illustrate this, consider a system where all machines are identical with $r = .1$, $p = .01$, and $\mu = 1$, and where all buffers have size 10. We let the number of machines in the system vary from 2 to 20, and set the loop population to half the total amount of buffer space in the system. By symmetry, the average buffer levels must then be 5, but the throughput of the system must be estimated. We did this by the two decompositions and simulation, as shown in figure 7-4. Since all processing rates are equal, the BDDX is in this case identical to the decomposition used by Frein, Commault, and Dallery (1994).

On this series of systems, the BDDX always converged quickly, while the ADDX decomposition failed to converge in one case and took up to 95 times more two-machine evaluations on the others. This was a surprising reversal of the usual reliability ordering between the decompositions. Both decompositions estimate a throughput of .7053 for all system sizes, while the simulations show throughputs ranging from .8590 for the two-machine system to .6777 for the twenty-machine system. This shows how the correlation effects cause the decompositions to underestimate throughput for small systems. As the system gets larger, these correlations become less important, and the decompositions approach the same degree of throughput overestimate as in a tandem line. As expected, both decompositions report average buffer levels of exactly 5.0, and the simulation reports average buffer levels of 5.0 $\pm$ .02.

7.2.3 Machines not identical

We now study a different, and perhaps more realistic, series of cases, based on the F series in Le Bihan (1994). These are tandem lines, but we modify them to closed loops by adding an infinite buffer leading from the last to the first machine of the tandem line. We then set the loop population, somewhat arbitrarily, to 30. The other line parameters can be found in tables 7.1 through 7.10. In tables 7.18 to 7.27, we again compare the results of continuous material simulations with the results from the two loop decompositions.

The column marked "j" indicates the search variable, which was $I_u(j)$. We tried all possible choices ($j \in \{1, \ldots, 5\}$) for the search variable. Combined with the 10 cases in the series, this gave 50 runs of the decompositions. The ADDX converged in all 50 cases, while the BDDX only converged in 29 cases. When multiple choices of j led to convergence, we report the results from the choice that required the fewest two-machine evaluations. These multiple choices led to the same values of the performance measures. As before, the number of two-machine evaluations is reported in the column marked "evals". Cases where an algorithm did not converge are indicated by blank rows in the tables.

The simulation experiments followed the same setup as before. Confidence intervals are omitted from the tables due to space considerations, but were calculated from the data. On the production rates, the 95% confidence intervals are about ± 0.001, and on the buffer levels ± 0.05 for all $\bar{n}_i$, except $\bar{n}_5$ where it is ± 0.1.

It is evident that the convergence behavior is not a strong point of the BDDX in these cases, since it failed to converge for four of the ten systems. Both decompositions tend to

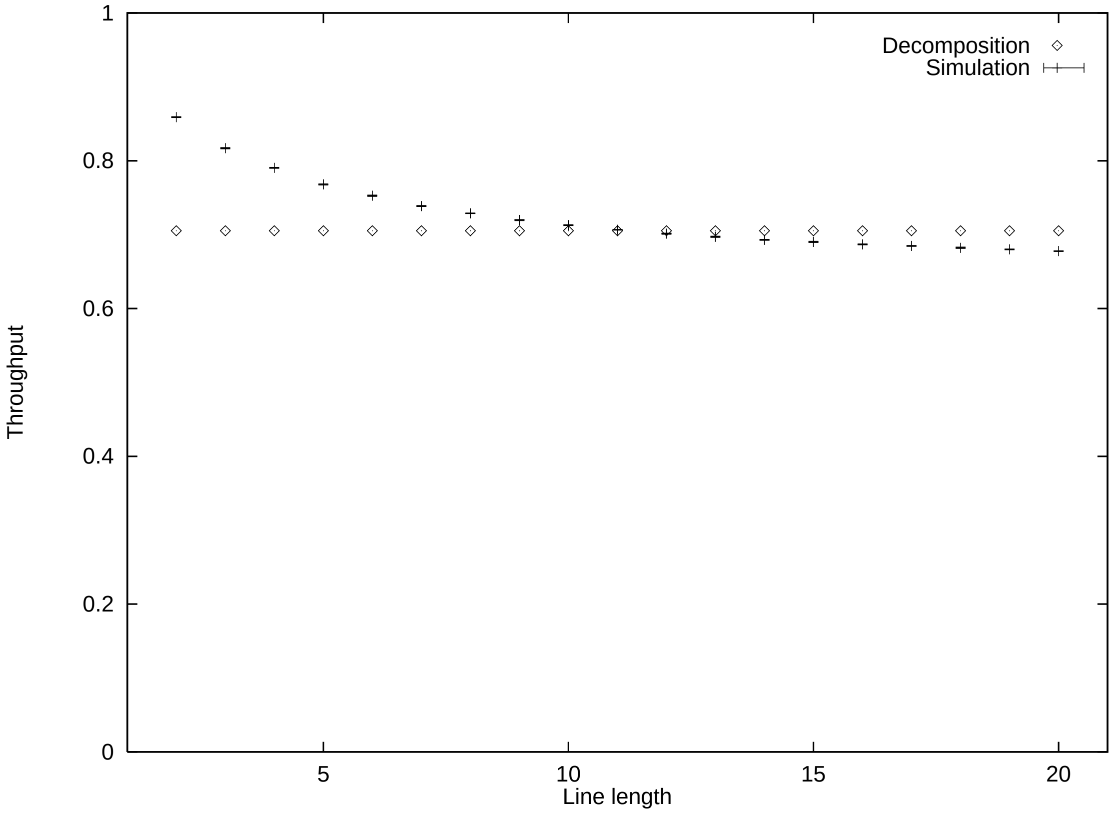

Figure 7-4: Throughput of closed loops of different lengths

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.854	5.08	16.60	1.16	2.56	4.60		
ADDX	0.850	3.78	16.35	1.03	1.98	6.85	1	5472
BDDX	0.868	3.63	18.00	1.08	2.09	5.20	2	28432

Table 7.18: Case LF1, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.854	4.88	12.15	8.69	1.31	2.97		
ADDX	0.845	6.05	8.73	8.48	1.27	5.46	2	2936
BDDX								

Table 7.19: Case LF2, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.850	2.79	9.60	4.77	11.45	1.38		
ADDX	0.851	3.56	10.27	2.33	11.80	2.04	3	6144
BDDX								

Table 7.20: Case LF3, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.880	1.23	5.25	4.59	1.04	17.89		
ADDX	0.840	1.22	3.38	2.95	0.94	21.50	1	1304
BDDX	0.840	1.21	3.21	2.89	0.94	21.75	1	1776

Table 7.21: Case LF4, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.866	11.16	2.70	3.71	6.39	6.04		
ADDX	0.853	11.84	2.40	2.23	5.28	8.25	1	5000
BDDX								

Table 7.22: Case LF5, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.831	4.14	14.37	2.06	3.88	5.54		
ADDX	0.817	3.65	12.85	1.74	2.97	8.78	1	2872
BDDX	0.828	3.52	13.30	1.80	3.10	8.27	1	2264

Table 7.23: Case LF6, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.835	5.36	10.07	7.66	2.40	4.50		
ADDX	0.818	5.79	6.93	6.87	2.19	8.21	2	2584
BDDX	0.821	5.71	6.61	6.97	2.23	8.48	1	2664

Table 7.24: Case LF7, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.828	4.05	10.65	3.78	9.07	2.45		
ADDX	0.819	4.96	10.16	2.24	8.15	4.49	3	2960
BDDX	0.829	4.99	10.12	2.11	8.24	4.54	3	2904

Table 7.25: Case LF8, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.847	2.25	8.20	5.00	2.13	12.42		
ADDX	0.816	2.42	6.29	3.77	1.83	15.69	1	1464
BDDX	0.816	2.40	6.24	3.75	1.83	15.79	3	1904

Table 7.26: Case LF9, loop population 30

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	j	evals
Csim	0.840	8.64	5.44	4.85	6.66	4.40		
ADDX	0.823	8.99	4.98	3.44	5.53	7.06	2	2960
BDDX	0.824	9.09	5.13	3.51	5.57	6.70	2	3232

Table 7.27: Case LF10, loop population 30

underestimate the throughput for these systems, and overestimate the average level of the infinite buffer. In the cases where the BDDX converged, it gave better throughput estimates than the ADDX.

7.3 Kanban

Our test cases for the kanban lines are again based on the F series of inhomogeneous lines in Le Bihan (1994). For all cases, we have added a finished goods buffer of size 30 and a demand process. The demand process acts like a machine with $r_D = 0.1$ and $p_D = 0.01$. We then used four different processing rates μ_D for the demand "machine". These rates were calculated to give average demand rates of 50, 80, 90, and 95% of the average throughput from the corresponding tandem line as determined by the continuous material simulations in section 7.1. We allowed unlimited backlogging of demand.

We solved each system by the equations derived in this thesis (key BDDX in the results) and by a decomposition combining Burman's ADDX equations for the tandem part of the line with our equations for the synchronization stage (key ADDX). We also made a continuous material simulation of each line. We ran 15 replications of each simulation case, each of 10^6 time units with an initial warm-up period of 10^4 time units, as in the previous sections of this chapter. The 95% confidence intervals for simulated production rates are $\pm$ .001, for the average levels of buffers 1 through 4 between $\pm$ .01 and $\pm$ .08, for the finished goods buffer $\pm$ .01 to $\pm$ .3, and for the system backlog $\pm$ .001 to $\pm$ 1.4. The wider confidence intervals were generated by the higher utilizations.

Tables 7.28 through 7.31 show the results for the first line. The other nine cases are similar. The column marked $\bar{n}_5$ indicates the average amount of finished goods waiting for delivery, and the column marked $\bar{n}_6$ is the average amount of backlogged demand waiting for products. In all cases that converged, the decompositions calculated the correct production rates. When the system utilization is low, the decompositions have excellent accuracy also in average buffer level estimates, as seen in table 7.28, and convergence is extremely quick.

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	evals
Csim	0.428	14.80	19.71	9.63	14.76	29.59	0.00	
ADDX	0.428	14.89	19.73	9.77	14.80	29.60	0.00	35
BDDX	0.428	14.89	19.73	9.77	14.80	29.60	0.00	35

Table 7.28: Case KF1.50, $\mu_D = 0.4711$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	evals
Csim	0.686	13.85	19.23	7.33	13.25	27.69	0.23	
ADDX	0.685	14.36	19.32	8.54	14.28	28.27	0.10	63
BDDX	0.685	14.36	19.32	8.54	14.28	28.27	0.10	63

Table 7.29: Case KF1.80, $\mu_D = 0.7538$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	evals
Csim	0.771	13.13	18.98	4.98	10.64	24.16	1.87	
ADDX	0.771	13.83	19.14	6.17	13.69	27.19	0.27	147
BDDX								

Table 7.30: Case KF1.90, $\mu_D = 0.8481$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	evals
Csim	0.814	12.67	18.83	3.25	7.66	17.97	9.62	
ADDX	0.814	13.59	19.05	2.76	11.96	26.03	0.48	336
BDDX	0.814	13.58	19.05	4.25	14.15	26.82	0.40	161

Table 7.31: Case KF1.95, $\mu_D = 0.8952$

The ADDX converged in all 40 of these cases (10 lines times 4 demand rates), and the BDDX in 38.

As the system utilization grows, both decompositions develop serious inaccuracies in the estimate of average system backlog, and to a lesser extent in the average finished goods level and other buffer levels towards the end of the line. Both decompositions tend to underestimate the system backlog ($\bar{n}_6$) and overestimate the finished goods level ($\bar{n}_5$). This gives an overly optimistic estimate of the system performance. As seen in table 7.31, these errors can become large. Similar errors result when analyzing homogeneous systems.

We believe there are two related causes for this behavior. Recall that in section 7.1, we found that the decompositions overestimated the throughput of the tandem lines. In case F1 (table 7.1), the throughput from the continuous material simulation was .857, and both the ADDX and BDDX decompositions estimated it to .862. This means that a demand rate that causes 95% utilization relative to the simulation result is only a utilization of 94% relative to the decomposition. Queuing theoretic formulas for average queue length generally include a term $(1-\rho)^{-1}$, where ρ is the utilization or traffic intensity (Kleinrock 1975). This difference becomes important as the utilization increases. For instance, $(1-.94)^{-1} = 16.7$, while $(1-.95)^{-1} = 20$.

However, this does not seem enough to explain the magnitude of this error. We believe that a deeper explanation can be found in the system variability. Essentially, the inhomogeneous decompositions work by aggregating the different processing rates that a particular machine can work at to a single average rate, and by replacing the unknown distributions of periods with and without flow into (and out of) each buffer by less variable exponentially distributed up- and downtimes of the machine upstream (and downstream) of the buffer. This preserves the means of the various distributions, but underestimates their variances.

In heavy traffic (high utilization) this can become very important. For instance, one approximate expression for the average waiting time in a G/G/1 queue is (Bertsimas and Mourtzinou (1994), eqn 3.23)

$$E[W] = \frac{\rho^2(c_x^2+1) - \rho(1-c_a^2)}{2\lambda(1-\rho)} \tag{7.2}$$

where ρ is the traffic intensity, λ the arrival rate, c_x the service time coefficient of variability (CV), and c_a the inter-arrival time CV. The CV is defined as the standard deviation of the distribution divided by the mean, and is a dimensionless measure of variability. This means that the average waiting time of this queue grows as the square of the CVs of both the arrival and service processes. By Little's Law (Little 1961)

$$E[L] = \lambda E[W]$$

the same is true for the average queue length. It has also been shown by Powell and Pyke (1994) that differences in the higher moments (skew and kurtosis) of the operation time distributions can cause significant differences (on the order of 5%) in the throughput of manufacturing systems.

These results are not directly applicable to the continuous material systems we are studying, since these have no operation (or service) time distribution, and no inter-arrival time distribution. But it seems reasonable to guess that the second and higher moments of

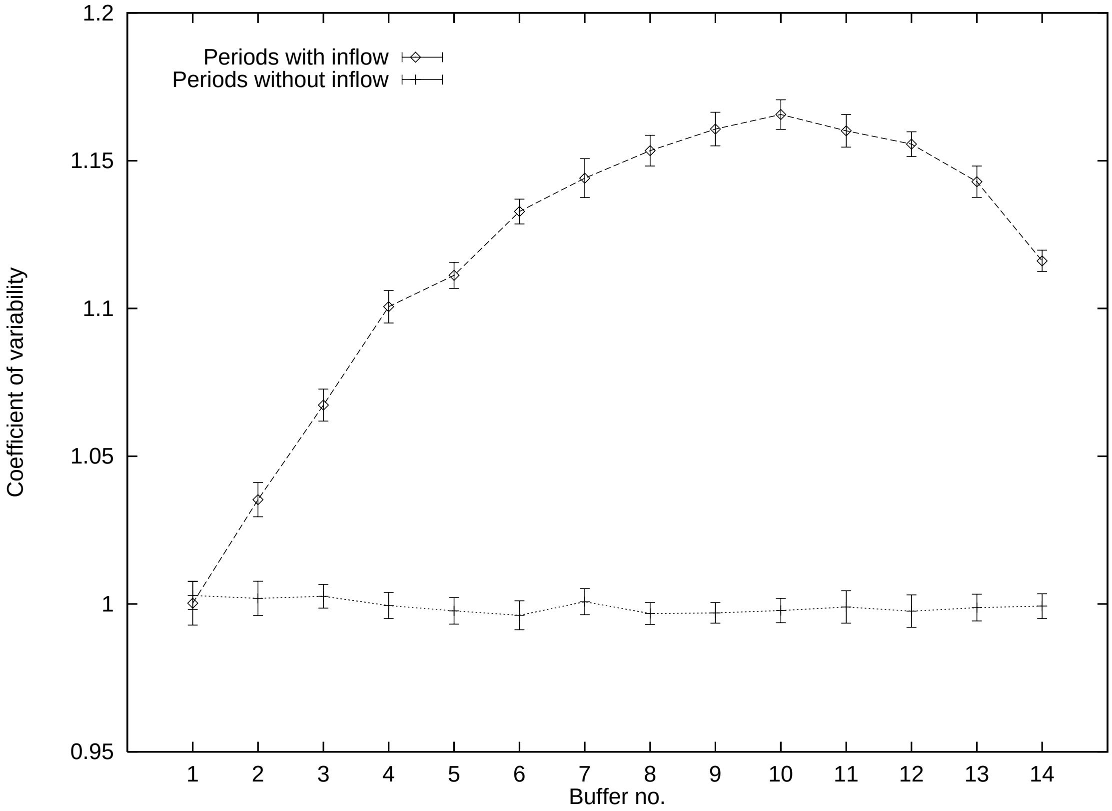

Figure 7-5: CV of periods with and without flow into the buffers of a tandem line

the up- and downtime distributions, as well as the variability of the processing rate, could have a similar influence on the average system backlog estimate.

We have done some very preliminary simulation studies of this effect, by measuring the duration of periods with and without flow into and out of each buffer of a 15 machine tandem line. All machines were identical, with $r = .1$, $p = .01$, and $\mu = 1.0$. All buffers had $N = 10$. With these parameters, we would expect the duration of periods without flow to be exponentially distributed with rate .1, since it is a mix of all the identically distributed exponential repair times upstream of each buffer. This then gives a CV of 1.0. The durations of periods with flow into the buffers are more complicated. They are a mix of the exponentially distributed times to fail of the individual machines upstream of the buffer and the non-exponential times to drain all buffers between the failed machine and the buffer where the observation takes place. As figure 7-5 shows, this gives a higher variability than the exponential distribution. The corresponding picture of the CV of times with and without flow *out of* the buffers is a mirror image of figure 7-5.

In lines synchronized to an external demand process, the CV does not seem to decrease towards the end of the line, but rises almost linearly all the way to the finished goods buffer. We have measured CVs of periods with flow into this buffer in the 1.2 - 1.4 range for the various control policies, which might be enough to cause the inaccuracies in the average backlog/finished goods estimates. This calls for further research, both in describing these variability measures, and in devising methods to capture their essentials in a decomposition.

7.4 CONWIP/kanban hybrid

In this section, we test the first of our hybrid policy decompositions, the one combining finite buffers with a single CONWIP-style loop population constraint. First, in section 7.4.1, we explore the behavior of the outer loop of the algorithm and its convergence properties. Thereafter, in section 7.4.2, we look at the accuracy of the results from the decomposition.

7.4.1 Convergence behavior

We expect more complications than in the loop case, since the interaction between the demand process and the line affects the estimated loop population. Recall that our search variable is $I_u(j) = p_u(j)/r_u(j)$ for some two-machine line j. For any machine in the line, if we increase the ratio p/r far enough, the machine has a lower production rate than the demand rate, and the system is unstable. We can therefore expect some unpleasantness around a value of the search variable $I_u(j)$ such that instability is about to occur.

Different choices of search variable

It turns out that the choice of the search variable index j is very important for the convergence properties of the algorithm. Poor choices of j lead to slow or no convergence. Figure 7-6 illustrates this for a 15-machine line where the demand rate is .5, and all machines have $r = .1$, $p = .01$, and $\mu = 1$. The population constraint (CONWIP limit) is 75, and the demand process is modeled as a machine with $r_D = p_D = .1$ and $\mu_D = 1$. All buffers have size 10. (These parameters are for testing only, and do not necessarily describe how to operate this system efficiently.) We have plotted curves of the estimated loop population $Q(I - u(j), j)$ as a function of the search variable $I - u(j)$ for three different choices of j. The curves were generated by a variation of the CONWIP/kanban hybrid decomposition of section 6.4.2, where the program stepped through a range of $I_u(j)$-values and performed the inner loop of the decomposition for each value. The unknowns were reinitialized to input parameters between calls to the inner loop procedure.

The most striking difference from the closed loop curves in figure 7-2 is the very steep drop around $I_u(j) = .96$. This is the onset of instability. That is, when $I_u(j)$ is above this point, the line throughput is less than the demand rate. Then the average backlog goes to infinity, and the inventory in the line is very low. For instance, if $I_u(j) = 1$, then $p_u(j) = r_u(j)$, so $e_u(j) = r_u(j)/(r_u(j) + p_u(j)) = 0.5$. Then the isolated production rate of that machine equals the demand rate, and the line as a whole cannot support the demand.

We can see from figure 7-6 that choosing $j = 8$ leads to much faster convergence of the search procedure for $I_u(j)$ than either $j = 1$ or $j = 14$. Computational results bear this out: Using $j = 8$, the algorithm performs 4,408 two-machine evaluations in about 0.4 seconds on a Sun SparcStation 10, while for $j = 1$ it takes 945,661 two-machine evaluations (67.0 seconds), and for $j = 14$ it takes 235,074 two-machine evaluations (16.8 seconds). Different values of the population constraint leads to different behavior, since this causes the

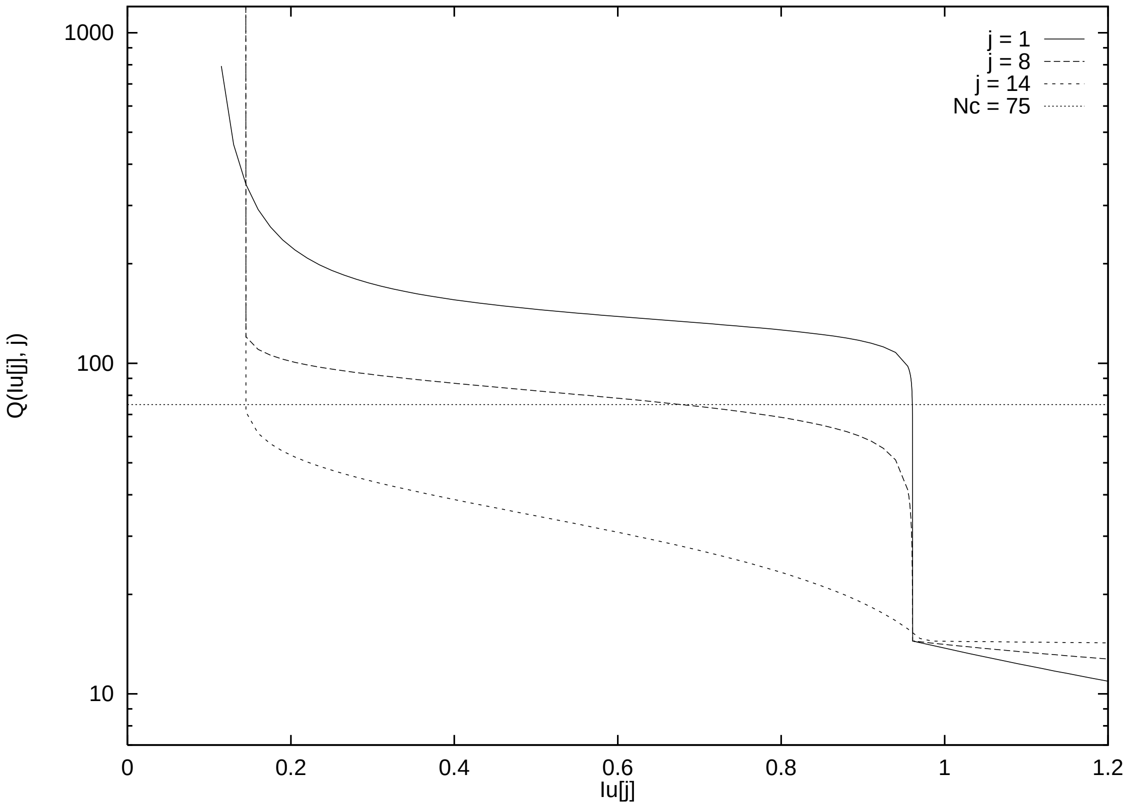

Figure 7-6: Behavior of $Q(I_u(j), j)$ for different j.

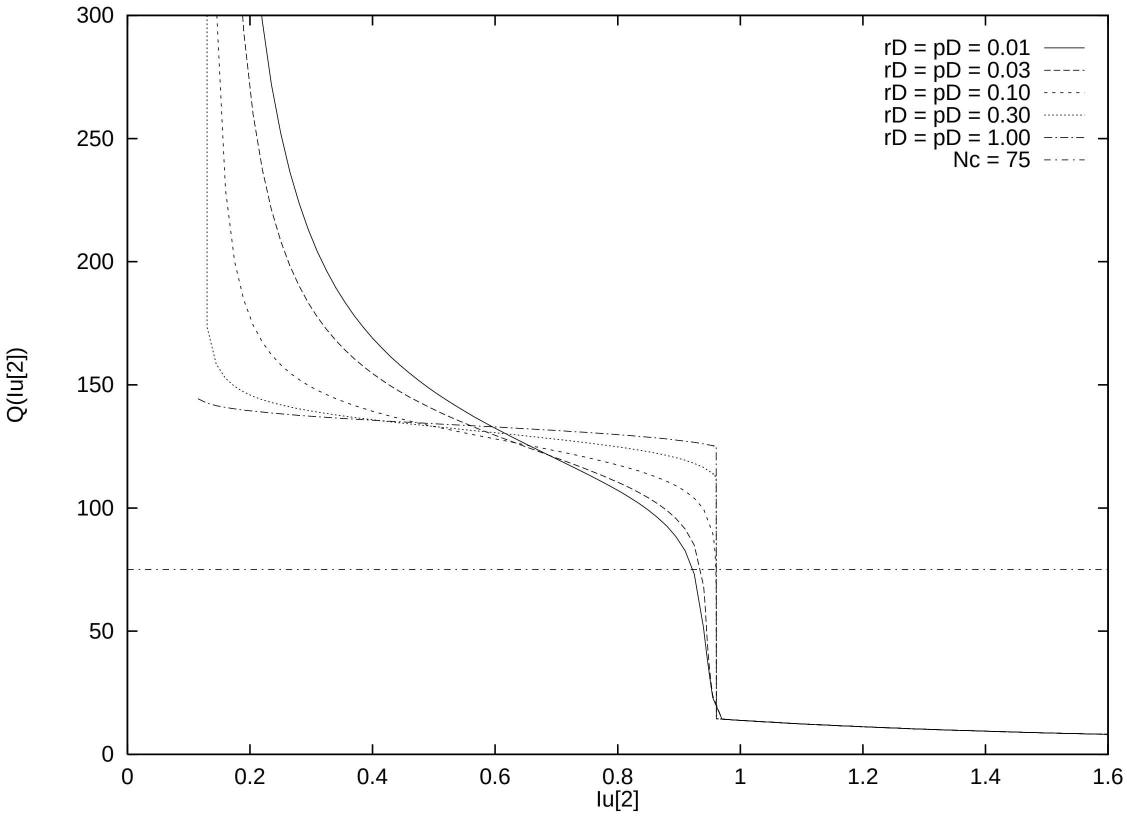

Figure 7-7: Behavior of $Q(I_u(2))$ for different demand representations.

horizontal line representing the population constraint to intersect the curves of $Q(I_u(j), j)$ in different locations. Similar behavior arises when the ADDX equations are used in the tandem part of the line.

Different demand process characteristics

The shape of $Q(I_u(j), j)$ depends on the system parameters. In figure 7-7, we have plotted this function for different representations of the demand machine in the same system as above, using $j = 2$. In all cases, the average demand rate is .5, but we change the failure and repair rates of the demand machine simultaneously, keeping $\mu_D = 1$. The function becomes steeper at the onset of instability as the failure and repair rates of the demand machine increase. We see that demand processes with very frequent short failures may lead to numerical problems in the algorithm, since in the extreme case, the curve for $Q(I_u(2))$ is little more than a horizontal line joining two vertical lines.

7.4.2 Decomposition accuracy

Different demand process characteristics

The parameters of the demand process affects the behavior of both the manufacturing system under study and the decomposition algorithm. From equation 7.2, we would expect changes in the variability of the demand process to have a major impact on the finished goods and backlog behavior of the system, even at a constant system utilization.

Consider a five-machine line, where all machines have $r = .1$, $p = .01$, and $\mu = 1.0$. We control this line by a hybrid CONWIP/finite buffer policy, where the loop population (CONWIP limit) is set to 20, and the internal buffers have size 10. As recommended in section 3.1, the finished goods buffer is equally large as the CONWIP limit. The buffer for detached cards going back to the first machine is unlimited. First, we need to find the behavior of this line in saturating demand. We do this by continuous material simulation of the system as a closed loop, and find a throughput of $.7683 \pm 0.001$. We note that the loop decompositions report a throughput of .693 for this line, an underestimate of 9.3%. We would expect this to lead to high estimates for system backlog and low estimates for finished goods levels in the hybrid case. Refer to figure 7-4 see how other choices of line length fare.

We fix an average demand rate of .6145. Compared to the simulated throughput in saturating demand, this gives about 80% system utilization. We now vary the parameters of the demand process while keeping the average demand rate constant. It is reasonable to expect that a more variable demand process gives more average backlog and less finished goods inventory than a less variable demand process. Our model of the demand process has three parameters: the processing rate μ_D, the failure rate p_D, and the repair rate r_D. If we hold the processing rate constant, the demand process becomes more variable if we decrease the failure and repair rates by the same factor. We can also make the demand process more variable by increasing the downtime $p_D/(p_D + r_D)$ while increasing the processing rate to maintain a constant average demand rate $\mu_D r_D/(r_D + p_D)$.

Base case Our base case has $r_D = 0.1$, $p_D = 0.01$, and $\mu_D = 0.676$, which makes the failure and repair process of the demand identical to that of the machines in the line. The demand rate is still a feasible .6145, since the demand process has a lower processing rate than the machines in the line. The results from applying our decomposition and the decomposition based on Mitchell Burman's ADDX equations to the base case are shown in table 7.32. The columns marked $\bar{n}_1$ through $\bar{n}_4$ are the average levels of internal buffers in the line, $\bar{n}_5$ is the finished goods buffer, and $\bar{n}_6$ is the average system backlog. Both decompositions overestimated the average system backlog, and underestimated the average finished goods level.

Long, infrequent demand interruptions To increase the demand process variability, we reduce the failure and repair rates of the demand process by a factor of ten, to $r_D = 0.01$ and $p_D = 0.001$, maintaining $\mu_D = 0.676$. This generates fewer, but longer disruptions of the demand stream. The results are shown in table 7.33. Compared to the base case, this increases the system backlog and increases the average inventory levels in the interior of the

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	.615	1.37	1.44	1.48	1.15	12.96	1.80		
ADDX	.615	1.35	1.52	1.76	2.87	9.93	8.12	5	1836
BDDX	.615	1.33	1.48	1.69	2.86	10.09	7.74	5	2840

Table 7.32: Case DV1 - base case

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	.615	1.61	1.71	1.74	1.32	11.74	3.07		
ADDX	.615	2.27	2.43	2.47	2.65	4.95	47.99	5	1755
BDDX	.615	1.77	1.98	2.28	3.24	7.06	23.71	5	3240

Table 7.33: Case DV2 - long and rare demand disruptions

system. The decompositions again overestimated the average backlog, but even worse than in the previous case.

Short, frequent demand interruptions In table 7.34 we have increased the failure and repair rates to $r_D = 1.01$ and $p_D = 0.101$, still with $\mu_D = 0.676$. This generates short, frequent disruptions of the demand process, and decreases its variability. Comparing to the base case, this decreases the system backlog and decreases the average levels of the internal buffers. In this case, the decompositions underestimate the backlog and overestimate the finished goods level. The transition from overestimating the backlog to underestimating it seems to take place when the demand failure and repair rates are between five and six times their values from the base case.

Short, intense bursts of demand In table 7.35 we have reported a case where the demand comes in short, intense bursts. The demand parameters are $r_D = .1$, $p_D = 10.0$, and $\mu_D = 62.1$. The increased processing rate compensates for the increased downtime to give the same average demand rate as before, .6145. This increases the backlog, decreases the finished goods level, and increases the internal buffer levels, compared to the base case. The decompositions again overestimate the average backlog and underestimate the average finished goods level.

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	.615	1.26	1.34	1.38	1.09	13.51	1.34		
ADDX	.615	0.88	0.85	0.74	1.21	14.64	0.56	5	1863
BDDX	.615	0.93	0.97	1.01	1.22	14.15	0.73	4	4890

Table 7.34: Case DV3 - short and frequent demand disruptions

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	.614	2.13	2.14	2.08	1.52	6.76	19.94		
ADDX	.615	2.07	2.19	2.40	2.76	3.39	76.98	5	1557
BDDX	.615	2.07	2.19	2.40	2.76	3.39	77.02	5	3180

Table 7.35: Case DV4 - short and intense bursts of demand

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	.615	1.26	1.33	1.38	1.08	13.54	1.32		
ADDX	.615	0.92	0.97	0.98	1.41	14.04	1.16	5	2268
BDDX	.615	1.06	1.13	1.15	2.46	12.23	3.67	5	2310

Table 7.36: Case DV5 - slow and reliable demand

Slow and steady demand In the last case in this series, table 7.36, we consider a case where the demand process is slow and reliable. This is represented by $r_D = 1$, $p_D = .00001$, and $\mu_D = .6146$. The backlog and inventory levels go down compared to the base case. The ADDX-based decomposition failed to converge for this case, and the BDDX again overestimated the average backlog, but less severely than in the base case.

7.4.3 Non-identical machines, varying utilization

We once more present results based on the F series of cases from Le Bihan (1994). As in the kanban experiments of section 7.3, we must consider the system utilization, but now the behavior of the system in saturating demand is given by the closed loop results of section 7.2 rather than the tandem line. We first look at the behavior for one value of the population constraint, then investigate the sensitivity to changes in the population constraint.

Population limit 30

We allocate a buffer of size 30 for finished goods inventory as in section 7.3, and set a loop population constraint of 30 as in section 7.2. We define a demand process that acts like a machine with $r_D = 0.1$ and $p_D = 0.01$. We use four different processing rates for the demand "machine," calculated to give average demand rates of 50, 80, 90, and 95% of the average throughput from the corresponding closed loop system as determined by the continuous material simulations in section 7.2. We allow unlimited backlogging of demand.

For each of these cases, we have run a continuous material simulation, and have performed decompositions based on the ADDX equations (with BDDX equations for the synchronization stage) and on BDDX equations alone. We tried using the upstream machine in each component as the one whose failure to repair rate ratio $I_u(j)$ is used for searching. This gave 200 runs of each decomposition. Of these, the ADDX-based decomposition converged in 154 cases, and the BDDX in 66.

We have presented the results from the first case in tables 7.37 through 7.40. As in section 7.3, the column marked $\bar{n}_5$ describes the estimated average finished goods inven-

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.429	0.26	0.38	0.22	0.41	28.57	0.03		
ADDX	0.429	0.23	0.25	0.22	1.55	27.60	0.02	1	97947
BDDX									

Table 7.37: Case CKF1.50, loop pop. 30, $\mu_D = 0.4716$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.686	0.77	3.29	0.72	1.44	22.86	1.07		
ADDX	0.686	0.56	2.52	0.82	3.29	21.96	1.40	5	2079
BDDX	0.686	0.55	2.33	0.74	2.44	23.14	0.79	4	7680

Table 7.38: Case CKF1.80, loop pop. 30, $\mu_D = 0.7546$

tory, and the column marked $\bar{n}_6$ is the estimated average backlog. The column marked "j" indicates which component was used for the search variable in the decomposition. Where multiple choices converged, we have reported the one that took the fewest component evaluations. Blank rows in the tables indicate cases that did not converge.

We see that the production rate estimates are very accurate, as expected, and that the buffer level estimates are accurate for moderate utilizations. For the higher utilizations, the decompositions overestimate the average finished goods level and seriously underestimate the average backlog. Even in these cases, the estimates of other buffer levels are reasonably accurate. The ADDX decomposition always estimates a higher average backlog than the BDDX. Unfortunately, the other cases evaluated in this series show that the backlog estimates do not always straddle the correct value as here. More often than not, both decompositions estimate too high backlogs and too low finished goods inventories.

Other values of the population limit

We have also tried the same series of experiments with loop populations of 40 and 20. If we increase the population to 40, the saturated throughput of the closed loop system increases from .854 to .856. Exposing this system to an average demand rate equal to 95% of the tandem line saturated throughput, we get the results in table 7.41. We see that both decompositions overestimate the finished goods level and underestimate the backlog. These accuracy of these estimates seem to be related to the accuracy of the throughput estimate

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.772	1.55	7.31	0.94	2.00	16.35	5.23		
ADDX	0.772	0.92	6.90	1.02	3.06	16.40	6.92	2	2133
BDDX									

Table 7.39: Case CKF1.90, loop pop. 30, $\mu_D = 0.8490$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.815	2.77	11.34	1.07	2.31	9.57	21.23		
ADDX	0.815	1.50	11.79	1.03	2.48	10.27	25.97	2	3690
BDDX	0.815	0.76	9.93	1.07	2.71	13.80	11.97	5	3190

Table 7.40: Case CKF1.95, loop pop. 30, $\mu_D = 0.8961$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.815	4.59	12.08	1.07	2.30	15.89	14.79		
ADDX	0.815	1.39	11.33	1.08	2.93	20.77	9.49	5	6075
BDDX	0.815	0.70	9.53	1.11	3.01	24.11	4.87	5	2970

Table 7.41: Case CKF1.95, loop pop. 40, $\mu_D = 0.8961$

of the underlying closed loop decomposition. In this case, an ADDX decomposition of the corresponding closed loop system (saturating demand) overestimated the throughput of the line. This is consistent with an overestimate of average finished goods and underestimate of average backlog in the hybrid case. Intuitively, the decomposition believes there is more capacity in the system than there actually is.

If we reduce the loop population to 20, the saturated throughput of the closed loop system decreases from .854 to .843. We repeated the previous series of experiments with a decreased CONWIP limit and the same demand rates. Table 7.42 shows the same line as before in a demand rate that corresponds to 95% of tandem saturated production rate. In this case, the decompositions underestimate the finished goods level and overestimate the backlog. This is the opposite tendency from when N_C was set to 40, and is typical of this series of experiments with $N_C = 20$. This is again consistent with the direction of the throughput error from the closed loop decompositions, since both decompositions now underestimated the throughput of the saturated system. The closed loop ADDX had the greater error with an estimate of .815 versus the BDDX estimate of .837.

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.815	0.74	10.00	0.84	2.13	3.89	42.20		
ADDX	0.815	1.60	11.97	0.95	1.78	0.05	7389.07	1	2322
BDDX	0.815	0.91	10.56	0.99	2.08	3.14	76.44	2	3200

Table 7.42: Case CKF1.95, loop pop. 20, $\mu_D = 0.8961$

i	r_i	p_i	μ_i	N_i
1	.15	.007	0.9	3
2	.1	.01	1.0	4
3	.1	.008	1.1	5
4	.2	.02	1.2	6
5	.1	.01	1.3	6
6	.1	.015	1.4	20
demand	.1	.01	0.6	∞

Table 7.43: The line that generated local minima

7.5 Basestock/kanban hybrid

7.5.1 Convergence behavior

The basestock/kanban hybrid decomposition makes a duplicate demand process that feeds machine 1, in addition to the one feeding demand into the synchronization stage. This is shown in figure 6-5. The buffer upstream of machine 1 is infinite, and the backlogged demand buffer may also be infinite. This calls for some care when searching over the failure to repair ratio $I_u(j)$ of some machine. Suppose this ratio is too high. Then the throughput of the line will be too low. On one end, the line is fed by the demand process, so the input buffer of machine 1 may go to infinity. On the other end, the demand process is putting backlogged demand into the backlog buffer, and its level may also go to infinity. However, the loop invariant, equation (6.20), is $Q_B = \sum_{i=0}^{k} \bar{x}_i - \bar{x}_B = Z_B$. If the first buffer becomes unstable, the estimated loop population Q_B goes to infinity. In a situation where the backlog is getting unstable, Q_B goes to minus infinity.

Figure 7-8 shows the estimated loop population $Q(I_u(j), j)$ for different choices of j. These data were generated by the BDDX basestock/kanban decomposition on a five-machine line where all machines were identical with $r_i = .1$, $p_i = .01$, and $\mu_i = 1$ for all i. The demand process had $r_D = .1$, $p_D = .1$, and $\mu_D = 1$. The hedging points of all machines were 20, which was also the size of the finished goods buffer. The internal buffers all had size 10, and the backlog buffer was infinite.

There are some new complications with this function. As in the CONWIP/kanban decomposition, it may be very steep in parts of its range. In this case, $Q(I_u(j), j)$ goes to minus infinity when the system becomes unstable. For low values of $I_u(j)$, the five curves in this figure intersect at $I_u(j) \approx 0.14$ and seem to be close to one another for values below that.

Another behavior of $Q(I_u(j), j)$ is illustrated in figure 7-9. Here it appears that $Q(I_u(5))$ has a local minimum, so there could be multiple solutions to the loop equation $Q(I_u(5)) = N_B$. This appears to be rare, and we have not pursued it further. This curve was generated by BDDX on the inhomogeneous system described in table 7.43, operated with a basestock level of $Z_B = 20$.

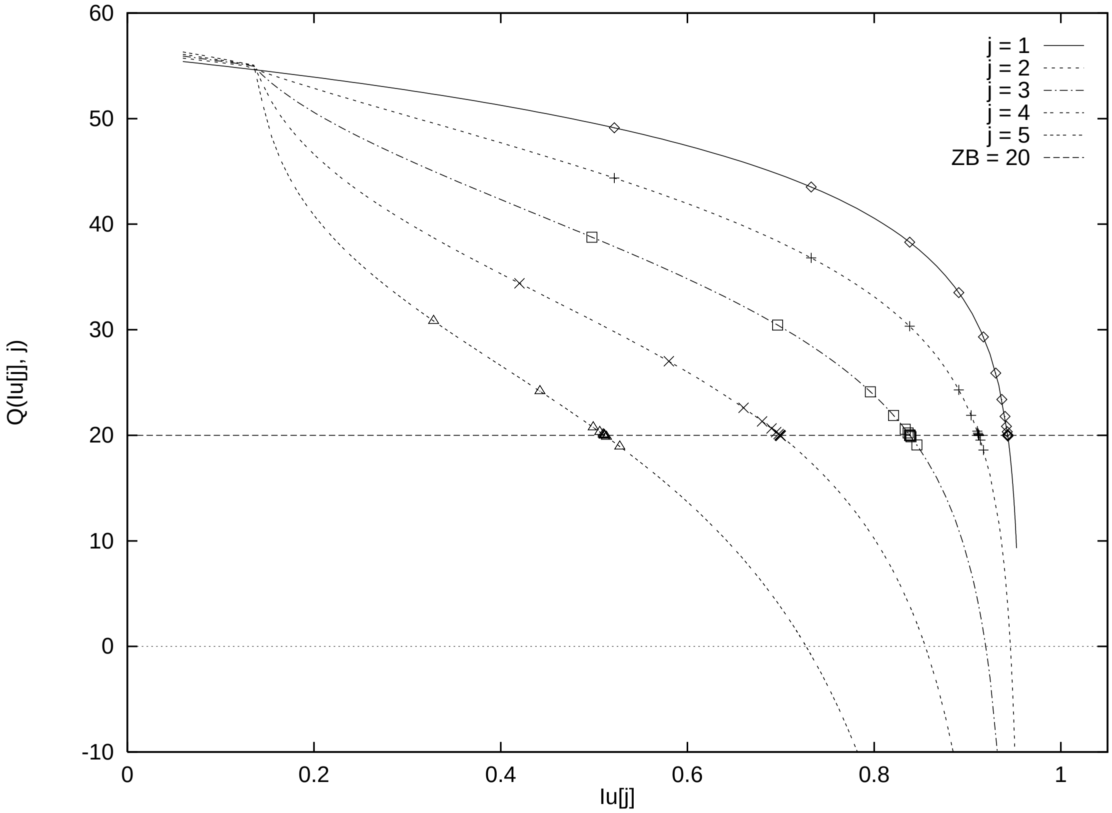

Figure 7-8: Behavior of $Q(I_u(j), j)$ for the basestock/kanban decomposition.

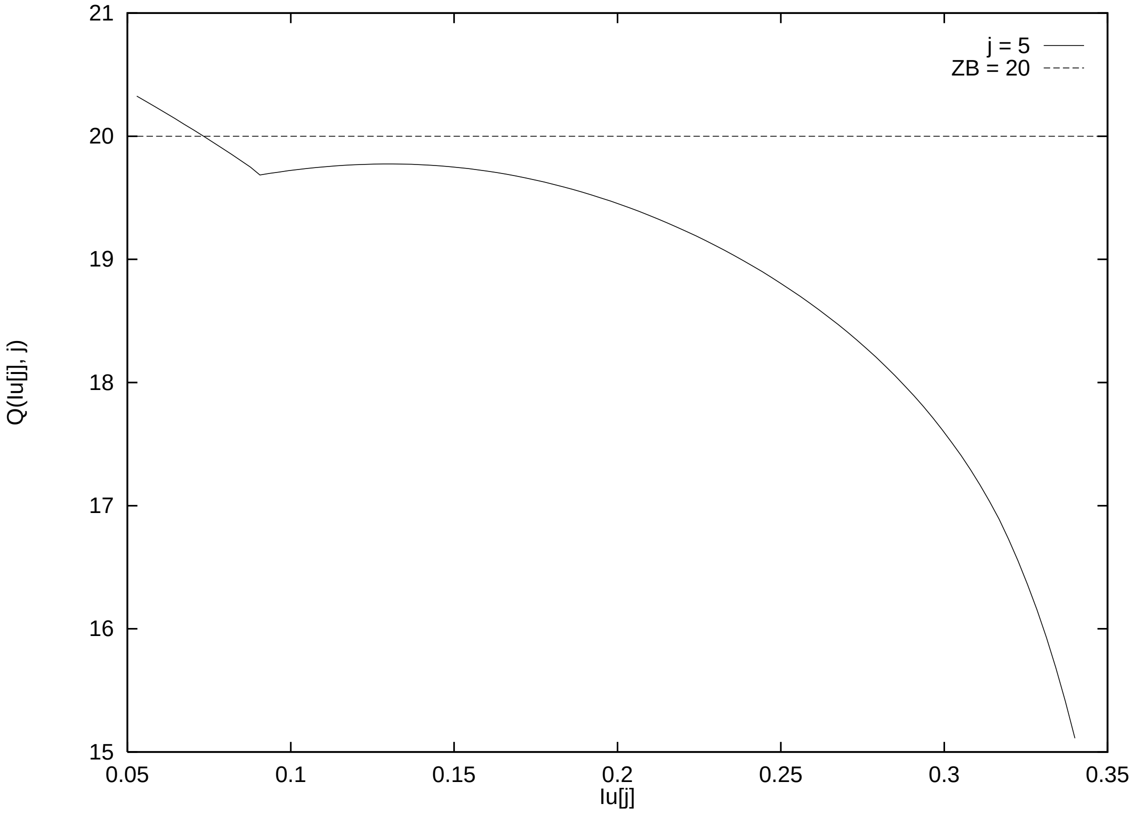

Figure 7-9: $Q(I_u(j), j)$ can have local minima

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.429	0.25	0.38	0.22	0.41	28.59	0.03		
ADDX									
BDDX	0.429	0.23	0.24	0.17	0.01	29.36	0.00	5	2460

Table 7.44: Case BKF1.50, loop pop. 30, $\mu_D = 0.4716$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.686	1.01	3.47	0.71	1.45	22.92	1.03		
ADDX	0.686	0.56	2.48	0.89	3.68	23.28	0.88	4	80360
BDDX									

Table 7.45: Case BKF1.80, loop pop. 30, $\mu_D = 0.7546$

7.5.2 Decomposition accuracy

We present one more series of experiments based on the F series of systems from Le Bihan (1994). The experiments are exactly like those done with CONWIP/kanban decomposition in section 7.4.3, but we now use the decomposition we have developed for the hybrid basestock/kanban control policy. For each of the ten lines in the F series, we added a finished goods buffer of size 30 (with a basestock level of 30) and set all other basestock levels to zero. We then used the same four demand processes as in section 7.4.

We ran simulation and decomposition experiments as before. Trying each search variable $I_u(j)$ for $s = 1, \ldots, 5$, on 10 lines with 4 loads, we again had 200 cases. The ADDX-based decomposition converged in 97 of these, and the BDDX in 34, so convergence was notably more difficult than in the other control policies. In tables 7.44 through 7.47 we have listed the results for the first line. These are again typical of the results we got in these experiments. We note that the decompositions get the production rate right, and give reasonably good estimates for the average levels of the internal buffers. The finished goods level is overestimated, and the backlog underestimated. This is consistent with the tandem line decompositions' overestimate of the saturated line throughput. This tendency is maintained when the basestock level changes to 20 or 40, and affects all converging cases in the series.

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.772	2.75	7.90	0.95	2.02	16.44	5.05		
ADDX	0.772	0.85	6.17	1.15	4.03	20.37	2.57	3	40830
BDDX	0.772	0.83	5.85	1.05	3.39	20.90	2.02	5	11100

Table 7.46: Case BKF1.90, loop pop. 30, $\mu_D = 0.8490$

Key	P	$\bar{n}_1$	$\bar{n}_2$	$\bar{n}_3$	$\bar{n}_4$	$\bar{n}_5$	$\bar{n}_6$	j	evals
Csim	0.815	5.64	12.28	1.07	2.32	10.02	18.34		
ADDX	0.815	1.29	10.67	1.21	3.79	18.00	4.95	5	26820
BDDX									

Table 7.47: Case BKF1.95, loop pop. 30, $\mu_D = 0.8961$

7.6 A longer line, revisited

In section 3.2 we simulated a ten machine line, where all machines had MTTF 100 and MTTR 10. The operation times were lognormal with mean 1.0 and CV 0.02, and the demand process was discrete with a probability of .5 of a demand occuring at each whole time unit. To assess the usefulness of the tools we have developed, we revisit this case.

First we need to approximate the discrete model by a continuous one. We keep the MTTR and MTTF of the machines, and get $r = .1$ and $p = .01$ for all machines. The operation times are nearly deterministic, so we approximate those as a constant $\mu = 1.0$. We keep all buffer sizes unchanged, which is not the only possibility. See Alvarez Vargas, Dallery, and David (1991b) for a discussion. The discrete demand process switches between a mode where it generates one unit of demand per unit of time and a mode where it generates no demand. There is a probability of .5 of going from one of these states to the other during a time unit. We therefore model it as a machine with $r_D = .5$, $p_D = .5$, and $\mu_D = 1$.

We can now compare each of our newly developed decompositions with the discrete material simulation from section 3.2, as shown in table 7.48. The performance measures are now the fraction of demand served from stock, the total work in progress in the line, and the average backlog. Estimates for these are easily calculated from the solution to the decompositions, as discussed in chapter 4. All decomposition results, marked "Deco" in the table, were generated by the BDDX equations for the appropriate control policy. The discrete material simulation results, marked "Dsim", are repeated from section 3.2 for ease of reference. We have also listed results from a continuous material simulator to see how much was lost in the transition from a discrete to a continuous model. These results are marked "Csim". Typical 95% confidence intervals for these are $\pm.0003$ for service level and $\pm.1$ for average WIP and backlog.

We see that the continuous material simulation predicts a higher level of service and a lower average backlog than the discrete model for all cases. This is surprising, since Alvarez Vargas, Dallery, and David (1991b) found that the continuous model of tandem lines with the same buffer sizes as in the corresponding discrete model tended to underestimate the throughput of the line. Since we used equal buffer sizes in our two simulation models, one might expect that the continuous model would give pessimistic estimates of service level and backlog, which is not the case.

The decompositions fall closer to the results from the continuous simulation than to those of the discrete simulation, but add their own approximation errors to the results. We note that in this case, the hybrid basestock/kanban decomposition had excellent agreement with the continuous material simulation, while the hybrid CONWIP/kanban decomposition nearly cancels out the approximation error introduced by moving to a continuous model.

Policy	Method	Service	WIP	Avg.backlog
Kanban	Dsim	0.757	26.4	4.9
	Csim	0.844	27.2	2.3
	Deco	0.911	28.7	1.0
Basestock/kanban	Dsim	0.751	19.5	2.2
	Csim	0.873	20.3	1.1
	Deco	0.873	21.3	1.3
CONWIP/kanban	Dsim	0.752	19.3	2.8
	Csim	0.819	17.2	1.9
	Deco	0.773	17.1	3.1

Table 7.48: Performance measures, 10-machine line

The decompositions rank the policies correctly with respect to WIP, but some of the service level and backlog estimates are so far off that this has limited value. The problem in comparing policies is again the throughput estimate of the underlying tandem or loop decomposition. The tandem decomposition tends to overestimate the throughput, so the kanban and basestock decompositions built on top of it err on the optimistic side. On the other hand, as seen in figure 7-4, the loop decomposition underestimates the throughput of small systems. In this case, it is close to the actual value, so the backlog estimate from the CONWIP decomposition is also good, but on the pessimistic side. This makes it difficult to use these decompositions for comparing control policies.

However, the decompositions took from .1 seconds (kanban) to 3.6 seconds (basestock/kanban) to run. In contrast, to achieve the accuracy reported in chapter 3 the discrete material simulation needed 30 replications of 240,000 time units each. This took about 20 minutes *when running on 10 workstations in parallel*, giving computational requirements a factor of 3,000 to 120,000 higher than that of the decompositions. If limited to a computation time of, say, three times the slowest decomposition, or about five months of simulated system operation, the discrete simulation tends to generate about as inaccurate numbers as the decompositions, but with random direction and magnitude of the error. The continuous material simulation is faster, and used about 70 minutes on a single CPU to complete 15 replications of 10^6 time units each. This is still about 1,200 times slower than the slowest decomposition.

This indicates that the decompositions can be valuable tools when computation time is at a premium, for instance when a large number of alternative parameter choices (such as kanban numbers or basestock levels) for a given system and control policy need to be assessed rapidly. This happens in system design or control policy design. The results can then be calibrated against a few simulation runs. We still cannot beat simulations when accuracy is paramount.

Chapter 8

Two-machine hybrid system in detail

In the previous chapter, we developed approximate methods for analysis of lines operated by a CONWIP/kanban hybrid control policy, as well as a basestock/kanban hybrid. We noted that these methods become inaccurate for small systems, since certain correlation effects are ignored by the decomposition methods. In this chapter, we explore one of the control policies, the basestock/kanban hybrid, in more detail to understand this phenomenon better.

8.1 Method

So far, we have studied a line with deterministic or nearly deterministic operation times through either direct discrete event simulation or through approximations that represent the material passing through the line as a fluid. We now study a different model, where the material is discrete, but the operation times are exponentially distributed. The demand is represented as a Poisson process, making the times between demand events exponentially distributed. We still assume that times to fail and repair are exponentially distributed. This leads to a system model that can be represented as a discrete state, continuous time Markov process. The models of chapter 4 through chapter 7 were all mixed state, continuous time Markov processes, where a part of the state was expressed through continuous variables and developed through differential equations.

With a discrete state Markov process, we can (at least in principle) enumerate all the states and assign individual probabilities to them. If the number of states is finite, this can be done by a matrix inversion technique. For some simple models, like M/M/1 queue, we can find a closed form solution to the state probability distribution. The case we study in this chapter is more complicated, and we resort to brute-force techniques.

In the following, we first present the model in section 8.1.1 and the solution principle in section 8.1.2. We then present numerical results, including a comparison of kanban and hybrid control, in section 8.2, before drawing some conclusions in section 8.3.

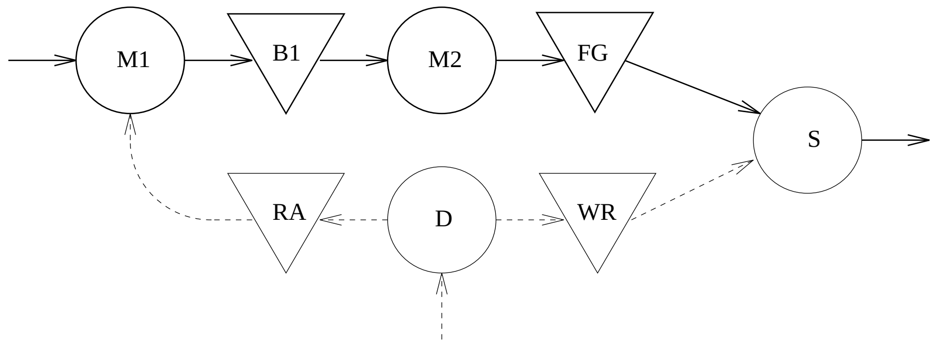

Figure 8-1: A representation of the system

8.1.1 Problem structure

We consider a two-machine tandem line, operated by the same hybrid basestock/kanban control policy as in section 6.5. Figure 8-1 shows the system using the same graphical conventions as in previous chapters. That is, solid arrows indicate movement of parts, dashed arrows movement of information. The machines M1 and M2 are separated by a finite buffer B1. Machine M2 has a finite output buffer FG for finished goods. The machines will not load another part if the immediate downstream buffer is full, i.e., the system operates by blocking before service. Machine failures are operation-dependent, so a machine cannot fail unless it is working on a part. Each machine can hold at most one part at a time.

A Poisson demand process D generates demand events that are instantly translated into a withdrawal request and a release authorization. The fictional synchronization machine S instantly joins an unit of finished goods with a withdrawal request and removes both from the system. If there are no finished goods, the withdrawal request is backlogged in buffer WR. The release authorizations generated by D are passed to machine M1. If the machine is performing an operation or failed, the release authorization waits in buffer RA. When machine M1 is available to do an operation, it must have a release authorization before it can load a part.

The parameters of the system are

μ_i, the processing rate for machine $i = 1, 2$
p_i, the failure rate for machine $i = 1, 2$
r_i, the repair rate for machine $i = 1, 2$
N_i, the buffer size for buffer $i = 1, 2$
d, the demand rate

In this model, the demand process does not fail, so in the notation of the previous chapters, the demand "machine" has $\mu_D = d$, $p_D = 0$, and $r_D = \infty$. Note that the Poisson demand process is still a source of significant variability, even without failures.

As in the analysis of the synchronization stage in section 4.3, buffers WR and FG cannot simultaneously be non-empty. We can therefore combine their levels into a single state variable x_2 that is the amount of finished goods minus backlogged demand. This variable measures the difference between cumulative production at machine M2 and cumulative demand, i.e., it is the *production surplus* of machine M2. We can now reinterpret the size N_2 of buffer FG as the *hedging point* Z_2 of machine M2, which is a target value of the production surplus that the machine will not exceed.

Let the state variable n_1 be the number of parts in buffer B1. Since, in this model, parts can stay in the machines for extended periods of time, we must account for any parts in the machines. Let m_1 and m_2 be the number of parts in machines M1 and M2, respectively. Let x_1 be the production surplus of machine M1. Note that there is a close relationship between the state variables n_1, x_1, m_2, and x_2, due to conservation of flow through the machines. We have that

$$x_1 = n_1 + m_2 + x_2 \tag{8.1}$$

Let Z_1 be the hedging point of machine 1. When both machines are at their hedging points, machine 2 will be empty, and buffer B1 will contain $S_1 = Z_1 - Z_2$ parts. This is the *basestock* level of buffer B1. (The basestock level of buffer B2 is Z_2.)

With an infinite buffer B1, this is a basestock control policy applied to the two-machine line (Kimball 1988). When combined with a finite buffer B1, this becomes a *two-boundary* control (Van Ryzin 1987). We have also referred to this control in sections 3.2 and 6.5 as a hybrid of basestock and kanban, using a kanban model equivalent to tandem queues (Berkley 1991).

Note that if we set the basestock level S_1 to the buffer size N_1, machine M1 will try to fill up buffer B1 whenever possible. The resulting control policy is kanban. This demonstrates that both basestock and kanban control are contained as special cases of this model.

We note that the number of waiting release authorizations n_0 in buffer RA is completely determined by the state variables we have discussed so far. Suppose both machines are at their hedging points. Then there are Z_2 parts in buffer FG and S_1 parts in buffer B1, no parts in the machines, no backlog, and no release authorizations. An arriving demand will add one release authorization and subtract one unit of finished goods. The release authorization causes a part to be loaded on machine M1, which later will be moved to buffer B1, loaded on machine M2, and end up in buffer FG. We have the system invariant

$$n_0 + m_1 + n_1 + m_2 + x_2 = Z_1 \tag{8.2}$$

Using equation (8.1), equation (8.2) can also be written

$$n_0 = Z_1 - x_1 - m_1 \tag{8.3}$$

This leads to a formulation with six random state variables:

α_i, the repair state of machine i for $i = 1, 2$
m_i, the number of pieces in machine i for $i = 1, 2$
n_1, the number of pieces in buffer 1
x_2, the production surplus of machine 2

For simplicity of notation, we define $n \equiv n_1 + m_2$ and $x \equiv x_2$. Note that we include the part in machine 2, if any, in the state variable n. This is because machine 2 will load a part whenever it is empty, its surplus is below Z_2, and there is a part in buffer B1. This instantly moves a part from B1 to M2, but does not affect the state in any other way. Similarly, M1

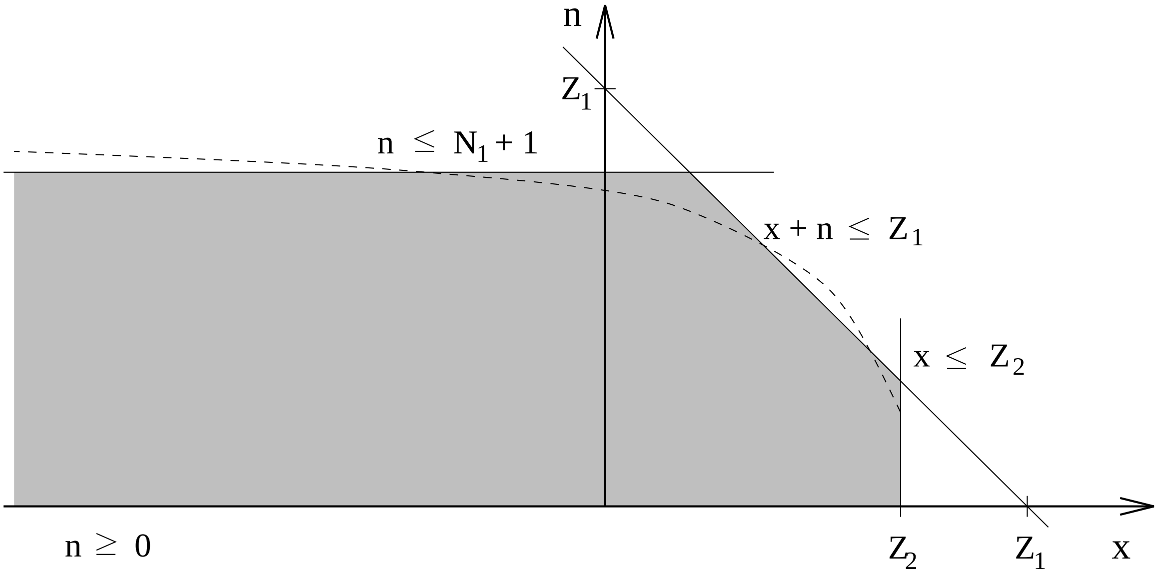

Figure 8-2: The attainable region in (x, n) state space

will load a part when it is empty, its surplus x_1 is below Z_1, and there is a space in buffer M1. From equation (8.1) we have that x_1 is determined by n and x. We conclude that the four state variables α_1, α_2, x, and n are sufficient to uniquely determine the system state.

In the two-dimensional space defined by the state variables x and n, the control boundaries of the system appear as straight lines. One control boundary is reached when the intermediate buffer fills up at $n = N_1 + 1$, another when the output buffer fills up at $x = Z_2$. A control boundary is also reached when the first machine reaches its hedging point at $x_1 = n + x = Z_1$. This boundary appears as a 45 degree line, because material processed at the first machine can be in the buffer or last machine, and therefore counted in n, or it may have left the system and be counted in x. In both cases, machine 1 holds back on production when this boundary is reached. A last boundary is $n = 0$, since we cannot have negative material in the intermediate buffer. Note, however, that the production surplus can take on negative values, so the region of attainable states is unlimited to the left, as indicated in figure 8-2.

In this figure, we have also indicated by dashed lines one possible shape of the optimal control policy as described by Van Ryzin (1987). The actual optimal policy depends on the parameters of the system, but this is a fairly typical shape. Note also that the hybrid control policy as drawn is a fairly close approximation to the optimal shape.

Figure 8-3 illustrates the transitions into and out of some state (x, n) away from the boundaries, and where both machines are up. The μ_1 transition is disabled when machine 1 is down, blocked, or at its hedging point. Similarly, the μ_2 transition is disabled if machine 2 is down, starved, or at its hedging point (i.e., the output buffer is full). We assume operation dependent failures, so a machine can fail whenever it is working. Equivalently, a machine cannot fail when it is down, at its hedging point, starved, or blocked. Blocking occurs before service, so a blocked machine does not contain a part. Failed machines do,

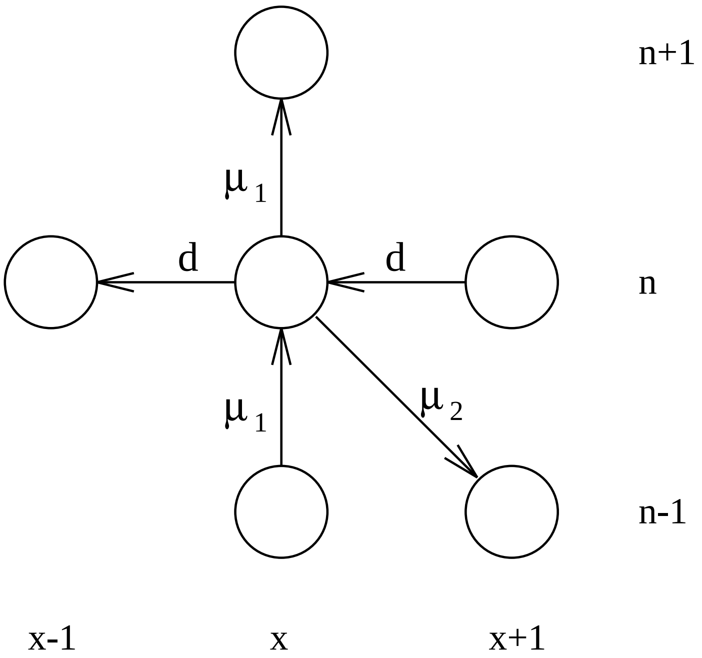

Figure 8-3: Transitions into and out of a state with both machines up

due to the operation dependent failure assumption.

8.1.2 Solution approach

In principle, we can find the steady state probability distribution of the system by solving the Markov process. This involves a matrix inversion, and, since we have an infinite number of states, the matrix will be infinitely large. We instead find a close approximation to the steady state probability distribution by assuming that the marginal distributions of x and n are asymptotically independent as x goes to minus infinity. We can then approximate the joint distribution for x and n in the far negative region of x by a product form solution. We get a truncated (and finite) Markov process describing the states not contained in the product form approximation. This Markov process can be solved. Joining this solution and the product form approximation, we have a complete solution.

Our solution approach consists of several steps:

- Find the buffer level distribution and production rate of the line in saturating demand.
- Check for demand feasibility.
- Construct a truncated Markov chain for the model, imposing the saturating buffer level distribution as a boundary condition along the truncation.
- Solve the Markov chain.
- Estimate the conditional buffer level distribution and the tail surplus distribution from the Markov solution.

- Use this to renormalize the Markov chain solution.
- Compute performance measures from the normalized solution.

In the following, we describe these steps in more detail.

Saturating demand analysis

If the demand rate is higher than the maximal sustainable throughput of the line, the surplus distribution does not exist, and the system backlog drifts to minus infinity over time. We refer to this situation as *saturating demand*. However, there is a buffer level distribution, since the machines still suffer failures and repairs, take unequal times to process parts, and so on. Under these conditions, the second machine is never blocked, and the line behaves identically to the exponential transfer line described by Gershwin and Berman (1981).

Our first step is to find this buffer level distribution. This can be done by the method of Gershwin and Berman or by solving the Markov process numerically, since it can be represented by a finite chain of $4(N_1 + 2)$ states. Let $\mathbf{p}(\alpha_1, \alpha_2, n)$ be the steady state probability of a particular state in saturating demand. From this distribution, we can find the maximal production rate of the system as

$$P = \mu_2 \sum_{\alpha_1=0}^{1} \sum_{n=1}^{N_1+1} \mathbf{p}(\alpha_1, 1, n)$$

This adds up the probabilities of all states where machine 2 is up and not starved, and multiplies the sum by the production rate of the second machine.

Feasibility check

The saturating demand production rate is used for a feasibility check on the given demand rate d. If the demand rate is higher than the saturating demand production rate P, no steady state surplus distribution can exist. In this case, we report the saturating demand production rate, the average buffer level, and the blocking and starvation probabilities of the line before terminating the program.

Truncated Markov chain

One straightforward way of approximating the system behavior when a surplus distribution exists is to truncate the Markov chain at some large negative surplus value X, and solve the resulting finite chain numerically. There are two difficulties with this approach: The truncation introduces distortions to the solution vector, and some of our performance measures (like average backlog) are strongly affected by the truncation. To avoid these problems, it is necessary to truncate at a value sufficiently negative to make the probability of being in the cut-off region negligible. For heavily utilized systems, this requires solving a very large Markov chain.

To work around these difficulties, we do two things: We impose the buffer level probability vector from the saturating demand analysis along the truncation line as an approximation to the actual solution to reduce the amount of distortion, and we approximate the distribution of surplus beyond the truncation point to estimate the performance measures better. Using these approximations, we get satisfactory results by solving a relatively small Markov chain. Figure 8-4 shows a simple Markov chain, with failure and repair transitions omitted for clarity, where the gray states have solutions imposed.

Solving the Markov chain

We solve the resulting system of linear equations using LU decomposition, done by a function in the Meschach library (Stewart and Leyk 1994). Note that the saturating demand buffer distribution is a probability vector, so it sums to one. Since we imposed these values at the truncation, our solution vector $\mathbf{q}$ has a sum greater than one, and needs to be renormalized to a probability vector. Before we can do this, we need to estimate the tail distribution of the production surplus.

Estimating the product form solution

As an approximation, we assume that the production surplus distribution is asymptotically independent of the buffer level and repair state of the system, and that it can be expressed as $\hat{p}(x) = a\rho^{z_2 - x}$ for some appropriate constants a and ρ, for x sufficiently negative. The value of ρ can be interpreted as the tail decay rate of the production surplus distribution. By analogy to an M/M/1 queue, ρ is likely to be close to 1.0 for a heavily utilized system and decrease with diminishing workload.

We estimate ρ from the solution vector $\mathbf{q}$ as follows: For each repair state (α_1, α_2) and backlog value x, we sum the solution vector over all buffer levels. This gives the marginal unnormalized surplus distribution $q(x, \alpha_1, \alpha_2)$ for each repair state. We now compute the ratio

$$\omega(x, \alpha_1, \alpha_2) = \frac{q(x, \alpha_1, \alpha_2)}{q(x+1, \alpha_1, \alpha_2)}$$

and a penalty

$$\pi(x) = \sum_{i=0}^{1} \sum_{j=0}^{1} (\omega(x, i, j) - \omega(x, j, i))^2$$

This penalty measures the squared error between the decay factors for the different repair states. Our estimate $\hat{\rho}$ is the value of $\omega(x, 1, 1)$ at the surplus value $\hat{x}$ that minimizes $\pi(x)$. Figure 8-5 shows how the $\omega(x, \alpha_1, \alpha_2)$ quickly approach each other for different repair states in a particular test case.

The conditional buffer distribution for any particular value of backlog in a stable system is different from that in saturating demand. To estimate the buffer level distribution beyond the production surplus truncation point, we select the unnormalized conditional buffer level distribution for $x = \hat{x}$ as an estimate of the conditional buffer level distribution in deep backlog. We discard the unnormalized probabilities from the Markov chain solution for surpluses less than $\hat{x}$ and replace these by the approximation

$$q(x, n, \alpha_1, \alpha_2) = q(\hat{x}, n, \alpha_1, \alpha_2)\hat{\rho}^{\hat{x} - x} \qquad \text{for } x < \hat{x}$$

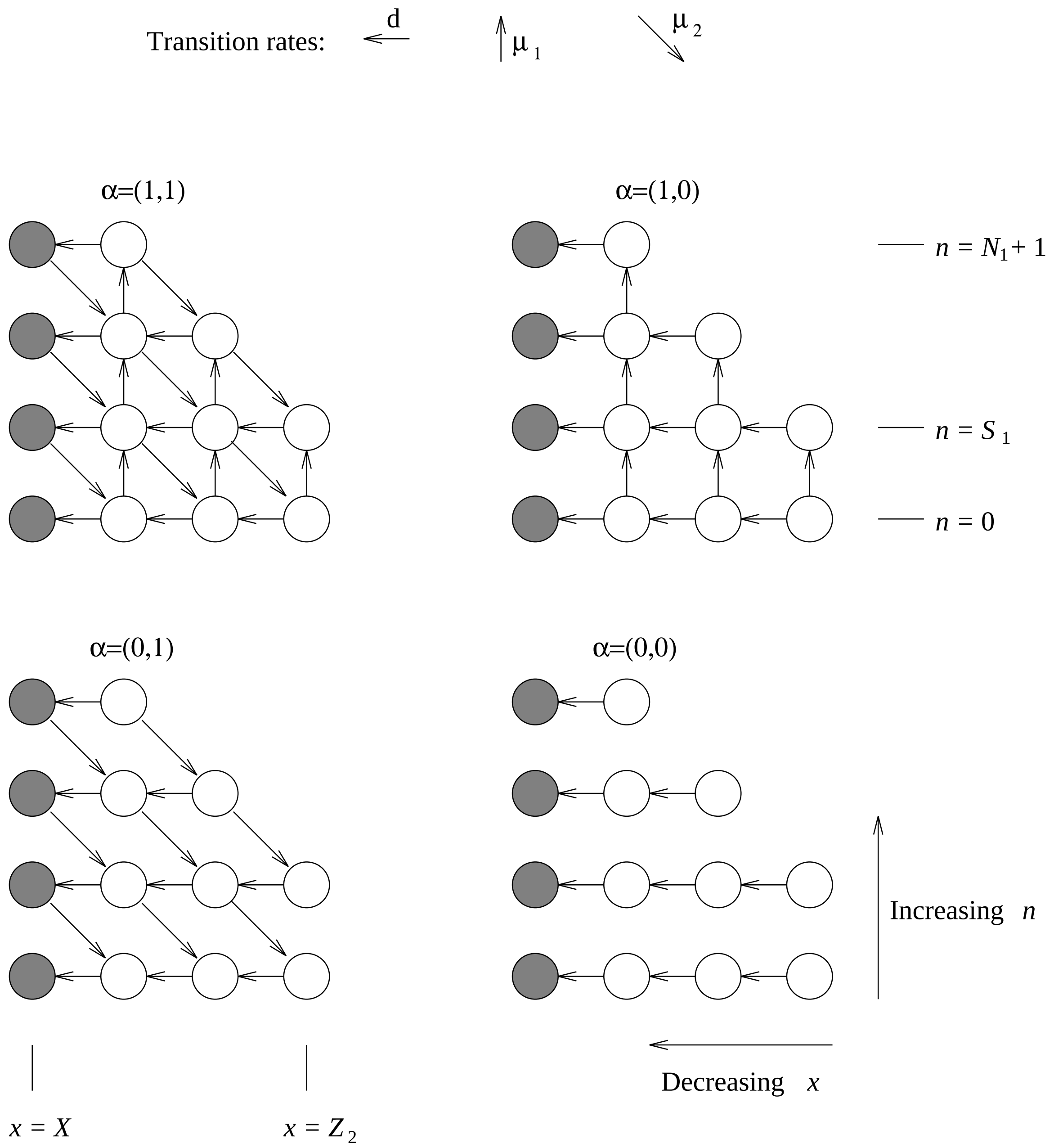

Figure 8-4: Truncated Markov chain with boundary condition imposed

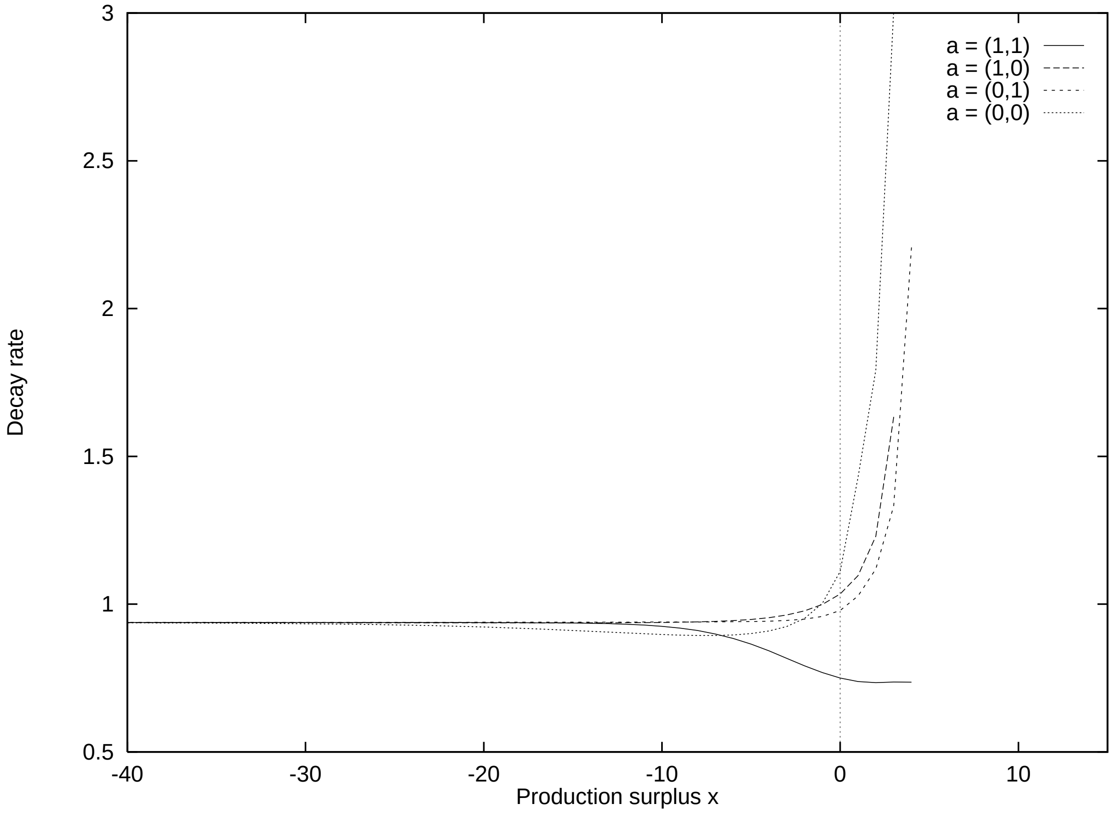

Figure 8-5: Marginal surplus distribution decay rates for the four repair states

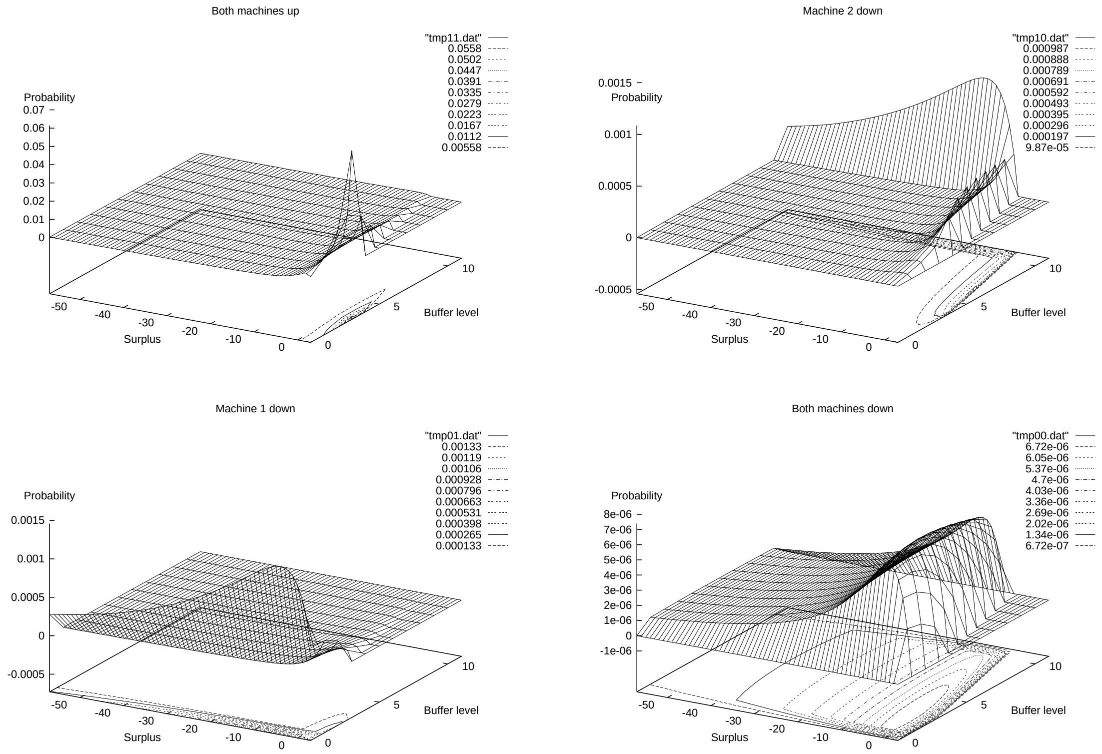

Figure 8-6: Markov solutions with approximated tail distribution

Renormalizing the distribution

To renormalize the distribution, we must find the unnormalized probabilities for the approximating patch. This depends on the sum of buffer level probabilities for the surplus value $\hat{x}$, since the approximation is a scaling of that vector. Let

$$a = \sum_{n=0}^{N_1+1} \sum_{\alpha_1=0}^{1} \sum_{\alpha_2=0}^{1} q(\hat{x}, n, \alpha_1, \alpha_2)$$

Then the total unnormalized probability in the region beyond $\hat{x}$ is the sum of a geometric series, and can be written

$$\sum_{x=\hat{x}}^{-\infty} \sum_{n=0}^{N_1+1} \sum_{\alpha_1=0}^{1} \sum_{\alpha_2=0}^{1} q(x, n, \alpha_1, \alpha_2) = \frac{a}{1-\hat{\rho}}$$

We can now renormalize the solution vector to get the steady state probability distribution of the system state. A sample distribution is shown in figure 8-6, where we have defined $p(x, N, \alpha_1, \alpha_2) = 0$ for $x + N > Z_1$. In this particular case, the probability distribution from surplus $\hat{x} = -45$ and down is approximated.

Looking at the distribution for $\alpha = (1,1)$, we note the spike at the joint hedging point (Z_2, S_1) and the ridge along the diagonal where $x_1 = Z_1$. When machine 2 is down, there is a concentration of probability along the same boundary, and a maximum on the boundary where $n = N_1 + 1$, some way away from the joint hedging point. When machine 1 is down, the behavior is similar, but along the boundaries $x = Z_2$ and $n = 0$. When both machines are down, there is a concentration of probability in the interior of the feasible region. For this case, the probability of being on a boundary is zero.

Calculating performance measures

Once we have the steady state probability distribution for the Markov process, we can compute the performance measures we are interested in for this line. These include the production rate, average levels in both buffers, fraction of demand served from stock, average customer waiting time, and the probabilities of blocking and starvation for the machines. All of these can be found by straightforward operations on the steady state probability vector.

8.2 Results

8.2.1 Correctness, accuracy, and computation time

If the demand rate is feasible, the production rate is regulated to match the demand rate. This makes the production rate calculated by our program an indicator of the solution accuracy, since we know the exact answer in advance. Other performance measures must be compared to simulations for accuracy. Since the accuracy of the solution depends on the size of the truncated Markov chain, and this in turn determines the computational effort, we also report the surplus value X for the initial truncation, the surplus value $\hat{x}$ used in the tail approximation, and the total probability included in the tail approximation.

We have run a large number of test cases, and tables 8.1 and 8.2 show the results from two representative cases. Table 8.1 represents a line in light utilization. System parameters are $p_1 = p_2 = 0.01$, $r_1 = r_2 = 0.1$, $\mu_1 = 1.6$, $\mu_2 = 1.5$, $N_1 = S_1 = 10$, and $Z_2 = 3$. The utilization factor reported is the ratio of demand rate over saturating demand production rate. For the simulation results, we ran 10 replications of 240,000 time units, each with an initial warm-up period of 9,600 time units.

For table 8.2, we have selected a case with the same machines and buffer sizes as in table 8.1, but we have set the basestock level to $S_1 = 2$ and increased the demand rate to 1.19. Even in this case, where 71% of the state probability is in the approximated tail distribution, we get the correct production rate to three decimal places, and all other performance measures within the 95% confidence intervals from the simulation.

We can get even better accuracy from the approximation by truncating the Markov chain at a more negative backlog value, at the cost of longer computation time. The algorithm used is $O(N^3)$ in the size N of the Markov chain, which grows as $N = O(N_1(Z_2 - X))$, where X is the surplus truncation point. We have truncated at $X = Z_1 - (2s+1)(N_1+2)$ where s is a user-controllable safety factor, initially set to 1.0. This leads to an algorithm that is $O(s^3 N_1^6)$.

	Analytic	Simulation
Saturating demand production rate	1.217	
Utilization factor	0.411	
Truncation point	$X = -23$	
Approximation transition point	$\hat{x} = -22$	
Probability in approximation	0.001	
Computation time	28.16 seconds	392 seconds
Production rate	0.500	$0.500 \pm .001$
Average internal buffer level	9.299	$9.305 \pm .011$
Average finished goods buffer level	2.379	$2.382 \pm .004$
Fraction served from stock	0.915	$0.916 \pm .002$
Average waiting time	0.611	$0.606 \pm .033$

Table 8.1: Test case 1: Lightly loaded asymmetrical line

	Analytic	Simulation
Saturating demand production rate	1.217	
Utilization factor	0.978	
Truncation point	$X = -31$	
Approximation transition point	$\hat{x} = -30$	
Probability in approximation	0.710	
Computation time	41.30 seconds	946 seconds
Production rate	1.190	$1.190 \pm .001$
Average internal buffer level	4.907	$4.925 \pm .045$
Average finished goods buffer level	0.100	$0.111 \pm .011$
Fraction served from stock	0.049	$0.054 \pm .005$
Average waiting time	91.93	79.63 ± 13.9

Table 8.2: Test case 2: Heavily loaded asymmetrical line

Policy	Prod.rate	From stock	Inventory	Avg.backlog
Best kanban	1.000	50.4%	7.83	14.64
Best hybrid	1.000	50.2%	6.46	9.55

Table 8.3: Performance measures, controlled two-machine line

A consequence of this is that our algorithm is not appropriate to estimate the performance of a pure basestock control, where the intermediate buffer size is unlimited. Any reasonable approximation would require a quite large buffer, which would lead to a very time-consuming computation. For this case, an algorithm like the one proposed by Lee and Zipkin (1992) would be better.

8.2.2 Kanban and hybrid control

In a kanban line, the machines are coordinated by blocking when their output buffers become full. Blocking occurs before service, so no new part is loaded by the machine before one is removed from the output buffer. The system behavior is regulated by the buffer sizes in the line, which can be changed by increasing or decreasing the number of kanban (cards) circulating in the line.

This can be seen as a special case of our hybrid control, where the basestock level S_1 of buffer B1 is the same as the buffer size N_1. That is, the first machine will always try to fill the intermediate buffer.

As an example, consider a problem where a demand rate of 1.0 is to be satisfied on a two-machine line with $r_i = 0.1$, $p_i = 0.01$, and $\mu_i = 1.5$ for $i = 1, 2$. We can adjust the line performance by changing the buffer sizes and the basestock level, and we seek the configuration that achieves a throughput of 1.0 with minimal inventory summed over the entire line.

We approach this by exhaustive enumeration of all line configurations with buffer sizes less than 5, and let the basestock level range from 0 to the size of the intermediate buffer. It turns out that the best choice is $N_1 = 3$, $S_1 = 0$, and $Z_2 = 1$, for a total line inventory of $\bar{m}_1 + \bar{m}_1 + \bar{m}_2 + \bar{n}_2 = 2.6255$. The best kanban policy used the same buffer sizes with 9% higher inventories. However, the service level is quite low, with only 2% of the demand satisfied from stock and an average customer waiting time of 53.5 time units.

To improve the service quality we impose the additional constraint that at least 50% of the demand has to be serviced from stock, and search over buffer sizes up to 10 for the intermediate buffer and 15 for the output buffer. The best hybrid policy has $N_1 = 9$, $S_1 = 0$, and $Z_2 = 8$, serving 50.2% of demand from stock at a total inventory of 6.46. With these parameters, the average backlog was 9.55. The best kanban policy has $N_1 = S_1 = 5$ and $Z_2 = 10$, which gives a total inventory of 7.83 and a service level from stock of 50.4%, with an average backlog of 14.64. These results are summarized in table 8.3. In other words, the two-boundary policy achieves the same production rate and service level as kanban, but at a 17.5% decrease in inventory and a 34.8% reduction in average backlog. This is consistent with our simulation results in chapter 3.

8.3 Discussion

The purpose of this chapter is to gain additional insight into the difficulties encountered by the hybrid basestock/kanban decomposition of section 6.5. Our decomposition approach for this control policy duplicated the demand process into one copy feeding the backlog/finished goods system, and one copy feeding the first machine in the line. This was shown in figure 6-5. These two "machines" are then assumed to be distinct and independent.

From the more detailed model we use in this chapter, we see that the number of release authorizations waiting for machine M1 is completely determined by the other state variables of the system.

Another source of decomposition error is that the decomposition assumes that the distribution of material in buffer 1 is independent of the distribution of system surplus. The buffer level distribution is then assumed to be of the same form as in a tandem line operating in saturating demand.

From studying figure 8-6, we note that the surplus and buffer level are strongly correlated when the system is operating close to its hedging point. This is evident from the spike at the hedging point, which distorts the distribution in the adjacent area. Assuming independence also requires the region where the distribution is defined to be rectangular. It is again clear that this is not the case for this system, since the hedging point of the first machine clips off a corner of the rectangle. Our solution procedure for the Markov process assumed that these two random variables are asymptotically independent as the backlog goes to minus infinity, and the accuracy of the solution seems to confirm this assumption.

Simply extending the solution method of this chapter to larger systems is not workable, since adding another machine and buffer increases the number of states by a factor of twice the buffer size to represent the repair state of the additional machine and the inventory state of the system. This will very quickly give a Markov process so large that it is intractable.

Future research may be able to devise improved decomposition methods, such as other building blocks than our two-machine line and synchronization stage, that allow these correlation phenomena to be represented more closely.

Chapter 9

Further research

Throughout the thesis, we have encountered items that were not fully resolved, and form opportunities for further research. These items are revisited in this chapter, and some promising research strategies are described.

9.1 Simulation studies

The simulation results in chapter 3 demonstrated the superiority of the hybrid CONWIP/kanban policy over kanban. We believe that this advantage grows with the size of the system, and with the process variability (such as machine failures and operation time variance). We also believe that the hybrid policy has an increasingly large advantage over CONWIP as the demand rate approaches system capacity. This should be explored further, and it seems that simulation is the most appropriate tool.

9.2 Quality issues

We have focused on systems where all the process variability is due to randomness in the timing of production, such as operation time variability and machine failures causing delays. Another interesting issue is raised when the variability is due to quality problems. One can identify two issues: Random production of bad parts (scrap) by a production process under statistical control; and a process that goes out of control, makes a lot of scrap, and needs to be stopped for remedial work. This is roughly analogous to operation time variability and machine failures in our quantity-oriented model.

Intuitively, our hybrid policies should outperform kanban also in systems where the variability is due to quality problems. The internal inventories of the system are kept low by a hybrid policy, so any quality problems will be identified quickly and fixed sooner.

This is a worthwile issue to explore, which is best done by simulation in an initial phase. When the problem is better understood, faster analytical models can be constructed. Eventually, it may be possible to combine the quality and quantity issues in a unified model.

9.3 Buffer size optimization

As we saw in section 3.2, it is not yet possible to optimize over the sizes of all buffers in a large system. It would be very valuable to do this, even in a purely academic setting, as the optimal choices of buffer sizes might give additional understanding of how the hybrid policies work and how they should be implemented in practice.

Such optimization consists of an evaluation subproblem and an optimization search. The subproblem is to evaluate the performance of a proposed set of control parameters for a particular line, and we have made some progress in this thesis towards solving this problem. What remains is the optimization search. At present, we do not understand how the performance of these control policies depend on the individual control parameters well enough to exploit any special structure that could simplify the search. This indicates that general purpose non-linear algorithms, possibly stochastic ones like simulated annealing and genetic algorithms, would be a reasonable first choice. Armed with additional understanding derived from this work, it might be possible to derive more sophisticated algorithms. One example of how this can be done is the buffer optimization procedure of Schor (1995) for tandem lines in saturating demand.

9.4 Applying hybrid controls to new systems

In section 3.1 we studied a 4-machine line without backlogging, in section 3.2 we studied a 10-machine line with backlogging, and in section 3.3 we took the leap to apply a hybrid control to a semiconductor factory with several hundred process steps, multiple part types, reentrant processes, and so on. It would be very interesting to study systems that exhibit one of these features at a time, to build more intuition about how the hybrid control policies work in a complex environment.

9.5 Systems with bottleneck machines

In the systems we studied in detail in sections 3.1 and 3.2 all machines were identical. Another case of practical interest is when one machine in the factory is a *bottleneck* that limits the overall capacity. Control approaches have been developed for this case, but these seem to assume either that the factory can sell everything it can make (Goldratt and Fox 1984), or that it is operating in a make-to-order mode (Fry, Cox, and Blackstone 1992). In the cases we have studied in this thesis, the production system is synchronized to a demand process, but the individual products are interchangeable.

One way of controlling a system with a bottleneck machine in an environment like this is to implement a hybrid CONWIP/kanban control with small buffer upstream of the bottleneck. These buffers would then fill up, providing a buffer inventory for the bottleneck, while blocking the release of parts to the system when the inventory upstream of the bottleneck reaches a preset limit. But this requires inventory to be present in buffers away from the bottleneck, where it cannot be used by the critical resource. Applying the same reasoning as in going from a local to a global information flow, we suggest that a better control policy circulates a set of cards between the bottleneck resource and the first machine in the pro-

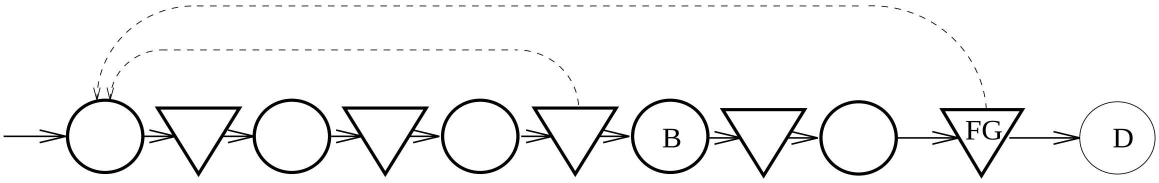

Figure 9-1: Proposed control for a system with a bottleneck

cess to limit this inventory, and another set of cards between the demand process and the first machine to synchronize the system to the demand process. In this method, the first machine in the system would need two cards to load a new part: One that indicates that a part has been consumed by the demand process (as in CONWIP), and one that indicates that the bottleneck machine has loaded a part. This is shown in figure 9-1, where machine B is a bottleneck.

This should be studied further and compared to other control approaches such as those studied in this thesis, as well as approaches developed explicitly for systems with a bottleneck.

9.6 Unequal inventory costs

In our simulation studies, we assumed that inventory costs were the same in all locations, i.e., we did not account for added value along the line. Some companies do calculate their inventory holding costs based on added value. For such systems, it might be advantageous to hold some inventory in an upstream buffer. Other reasons for holding inventory internal to the system could be perishability of the material in a certain part of the process, or a process that makes several final products after a number of common process steps.

Systems like these can be controlled by a hybrid basestock/kanban control, like the one shown in figure 2-1. The target inventories of the various buffers are set by the basestock part of the control, and the kanban cells again act as secondary controls that kick in to block release of material in the case of machine failure or similar problems in the line.

This should hold similar or greater advantages to kanban as the cases we studied, since we can now ensure that there are very low inventory levels in critical parts of the process, while the system has sufficient buffering to achieve throughput and service goals. This should be explored further, for instance in a simulation study of a specific production system.

9.7 Assembly/disassembly systems

It seems straightforward to extend the decompositon equations of chapter 5 to general tree-structured networks, using an approach similar to that of Gershwin (1991) and Di Mascolo, David, and Dallery (1991).

9.8 Assembly/loop production systems

One common type of production line is shown in figure 9-2. It consists of a single closed loop where a fixed number of pallets circulate. At one station, raw material is loaded onto the pallet. Going through a sequence of stations, the raw material is transformed and parts are assembled on to it until the product is finished. It is then detached from the pallet, and the empty pallet recirculates to the first station. Upstream of assembly operations, there could be a single machine feeding the assembly line through a buffer of parts, a tandem line, or a subassembly system as shown in the figure. This type of production system occurs for instance in the electronics industry, where components are assembled onto circuit boards.

A system like this would be tree-structured if it were not for the single loop. Tree-structured networks have been studied by Gershwin (1991), Di Mascolo, David, and Dallery (1991), Liu and Buzacott (1990), and others. It seems reasonable to try analyzing this class of systems through an approach similar to the one we have used for the hybrid systems. The only difference would be that instead of having a straight line decomposition as the inner loop, one would now have an assembly/disassembly decomposition.

A decomposition like this could be a very useful tool in industrial applications, and could, if combined with a sufficiently reliable and accurate set of decomposition equations, be of some commercial value.

A further goal is to develop performance analysis techniques for arbitrary manufacturing networks. This may require new building blocks, analogous to our two-machine lines and synchronization stages, new decomposition equations, and possibly new algorithms for solving the resulting system.

9.9 Generalized exponential decomposition

As the numerical results in section 7.1 demonstrated, the existing decompositions for inhomogeneous tandem lines have a tendency to overestimate the production rate of a line. Similarly, in section 7.2 we saw that the decompositions underestimated the throughput of some loop systems, and in section 7.4 the decompositions overestimated the backlog of the corresponding controlled system. This may be due to the way our decomposition equations match only the first moment of the unknown distributions. These distributions (for quantities such as the time with and without flow into and out of each buffer) are then replaced by less variable exponential distributions.

One promising decomposition approach that has not been explored in this thesis was developed by Dallery (1994). In this method, so far only applied to homogeneous systems, the downtimes of machines are represented by *generalized exponential distributions* consisting of an impulse at zero and an exponential tail distribution. This allows the decomposition to match two moments of the downtime distributions, instead of just one as in the approach we used. The outcome of this representation is then transformed into a machine with exponential downtimes, but where the average up- and downtimes are increased, adding variability to the system.

The analytic approach recommended by Le Bihan (1994) for slightly inhomogeneous lines is a transformation of the line to a homogeneous line, followed by this generalized

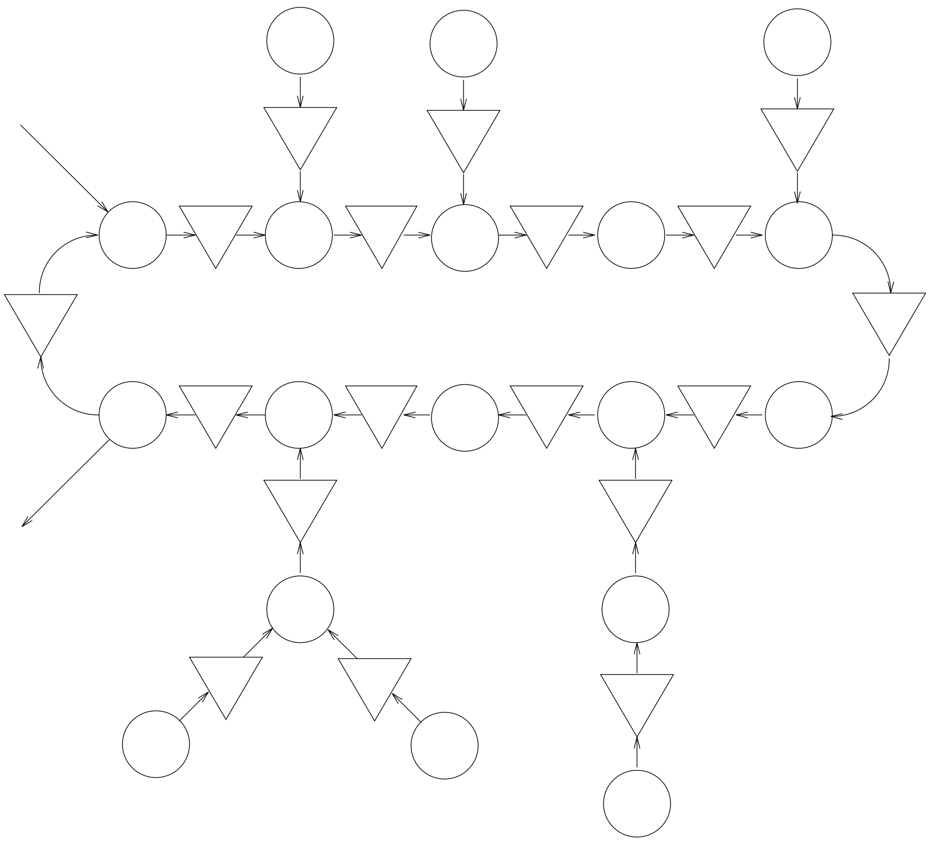

Figure 9-2: An assembly/loop system

exponential decomposition. This gives very good estimates of production rates for the lines described in section 7.1. For reasons explained in section 1.1.2, we did not use this approach.

Also, in personal communications, Dallery reports good results from using a generalized exponential decomposition on loop systems. It seems to have better convergence properties than the purely exponential decompositions. This could be a great advantage in the more complicated decompositions we have studied.

It is possible to develop an inhomogeneous extension of the generalized exponential decomposition. This could be done either as an extension of the decomposition procedure developed in this thesis, or as an extension of Burman's method. Developing such a decomposition for the various systems studied in this thesis seems to be a reasonable research project.

9.10 ADDX synchronization equations

In chapter 7, we saw that the ADDX equations give a more robust behavior than the BDDX equations. The decomposition equations developed in section 5.2 to describe how the synchronization stage influences the rest of the line are of the same form as the tandem BDDX equations.

It is possible to develop ADDX-style equations for the synchronization stage. The main work in doing this is to develop interruption of flow (IOF) equations, and use those instead of the processing rate equations in the BDDX. This is likely to give a more stable algorithm than either BDDX equations alone or combined ADDX/BDDX equations.

9.11 Interruption of flow variability

The existing generalized exponential decomposition matches two moments of the time to resume flow after an interruption. As we saw in chapter 7, the time from the flow is resumed until it is interrupted again has a higher variability, also when the machines are equal. This seems to be one of the major sources of approximation error in the backlog/finished goods estimates of our decompositions.

It would be very interesting to derive equations for the second moment of the time to flow interruption, combine this with the existing equations for the first moment (Burman 1995), and apply the generalized exponential distribution to get an equivalent machine with exponential up- and downtime distributions. This might be somewhat involved, but could make a big difference in the accuracy of the decompositions.

9.12 Multiple failure mode two-machine line

The two-machine lines studied in chapter 4 have a single exponentially distributed failure mode. One can envision alternate models with multiple failure modes, so each machine will have several failure rates, each leading from the working state to one of the failure modes. There will also be several repair rates, each one leading from one failure mode back to the working state. Taken as a whole, such a machine will have hyperexponentially distributed up- and downtimes. This model could yield a tractable solution. If there are m failure modes per machine, there are $2(m+1)$ discrete states for the line in addition to the single continuous buffer level variable. This gives a system of $2(m+1)$ coupled ordinary differential equations describing the steady state behavior of the system. To solve this system, we will have to solve a higher order polynomial than the quadratic we solved in section 4.2. This may not be tractable in closed form, but the roots of the polynomial are readily found by well-known numerical methods (Gerald and Wheatley 1994), leading to a solution for the steady state distribution of the buffer level. As before, the interesting performance measures for the two-machine line can then be calculated from this distribution.

Two-machine models like this could then be used as components in a decomposition. For instance, machines with two failure modes could be used to match two moments of the durations of periods with and without flow into (or out of) a buffer, in the same way as we now match the first moment of these durations with the single failure mode of the machines. This requires additional decomposition equations to be developed, and possibly a new algorithm for solving the resulting system of equations. This should be investigated, but may prove to be a hard problem.

One major advantage this procedure might realize over methods using generalized exponential distributions is that the higher moments of the resulting hyperexponential distributions would be closer to the corresponding quantities in the actual line. The generalized exponential distribution has an impulse at zero and an exponential tail, which leads to very different higher moments than what is generated by a mixture of exponential distributions. As shown by Powell and Pyke (1994), these higher moments could have a significant influence on the line behavior.

9.13 Decomposition convergence regions

The common solution procedure for decompositions is a fixed-point iteration, and different sets of decomposition equations lead to different convergence regions, as we have seen in chapter 7. There has not been any systematic effort to map the convergence behavior of these equations.

Even a very simple iteration scheme can give rise to extremely complicated convergence regions, as evidenced by the famous Mandelbrot set. This is nothing but a map of the convergence region for a very simple iterated equation in the complex plane.

It would be an interesting project to map the convergence regions for a number of decomposition approaches. This would require significant computational resources.

9.14 Improved basestock/kanban decomposition

The decomposition for the hybrid basestock/kanban control policy, described in section 6.5, has convergence difficulties. It would be interesting to study this further, and develop refinements to the decomposition equations and/or the solution algorithm to make the decomposition more stable.

9.15 Multiple basestock levels

We restricted our hybrid basestock/kanban control policy to cases where all basestock levels except finished goods are zero, or equivalently, where all machines have equal hedging points. This could be extended to systems with different hedging points. In this case, there is a separate information loop from the demand process to each machine. Conceptually, each arriving demand generates a release authorization that is passed to machine 1, a withdrawal request that is passed to finished goods, but also work orders that are sent to each machine 2, 3, ..., k. This allows the machines to maintain their production surpluses independently.

The resulting decomposition will have one loop invariant for each of the machines. This requires some method to handle multiple, nested search loops of the type we used in the loop and hybrid decompositions; or a completely new approach.

9.16 Continuous material hybrid two-machine line

In chapter 8 we analyzed a discrete material model of a two-machine line controlled by a hybrid basestock/kanban control. To do this, we used exponential distributions for the operation times and inter-demand times. These distributions have coefficients of variability of 1.0. This is more variability than many manufacturing systems exhibit. The simulations in section 3.1 used operation times with a CV of .02 and deterministic times between demand events (CV 0.0). In section 3.2, we had an inter-demand time CV of 1.0, but still operation time CV of .02. And in section 3.3, both operation times and inter-demand times were deterministic.

In the decompositions, we handled these low-variability operation times through a continuous material model. It would be very interesting to extend the approach of chapter 8 to a continuous material model. The state variables x and n then become continuous variables, and must be described through partial differential equations with boundary conditions. There will be a two-dimensional probability density in the interior of the feasible region (see figure 8-2), one-dimensional densities along the control boundaries, and a zero-dimensional impulse at the joint hedging point (Z_2, S_1).

We have developed the partial differential equations and boundary conditions that describe this system (Bonvik 1996). These equations can be solved numerically through a number of standard methods. See, for instance, Gerald and Wheatley (1994) for an introduction. In most methods, the state space is discretized and a related set of linear equations is derived. These linear equations are then solved through a standard technique. Their solution describes a close approximation to the continuous distribution, usually by interpolating linear (or low-order polynomial) functions between the discretization points.

Now that the partial differential equations and boundary conditions are already derived, working out a numerical procedure for solving them is a reasonable project for someone with some background in numerical computation.

Chapter 10

Conclusions

We have presented a new class of control policies for manufacturing systems. These policies are hybrids between existing policies, chosen to get a proper balance between global and local information flow in the factory. Our simulation studies show that the hybrid policies outperform other policies, such as kanban, minimal blocking, basestock, and CONWIP with respect to attaining high service and throughput goals at low inventories. The hybrid policies are card-based, and therefore retain the organizational learning benefits of kanban control. The performance advantage, the simplicity of the policies, and the ease of implementing them in the factory indicate that the hybrid policies hold great promise in practice.

We have developed fast, approximate decomposition methods for estimating the performance of systems operated by these policies under particular choices of the operating parameters. Novel elements of the decomposition methods are that multiple types of building blocks are used within the decomposition, and that both information flow and material flow are represented in the model. The resulting programs are very fast and reasonably accurate. This permits simultaneous design of a factory and its operating policies, without resorting to rules of thumb or time-consuming simulation studies.

Appendix A

Distributed discrete event simulation

Discrete event simulation is, almost by definition, a very CPU-intensive task. Some stochastic process is represented by a program incorporating a pseudo-random number generator, and statistics are collected by executing the program. It may be necessary to run the simulation for a very long time to get good estimates of the performance measures of the system under study. For example, if one is studying a manufacturing system where machine failures are important to the overall behavior of the system, the simulation must cover a period with many failures of different lengths before a good estimate of steady-state conditions is reached.

It is very hard to parallelize a discrete event simulation, because the commonly found algorithm (see Sedgewick (1992), chapter 11) maintains a centralized event list and a current simulation time, executing one event at a time. The current event may manipulate the list of future events, for instance by scheduling the end of some activity or the next instance of some event. When the current event is done, the event with the lowest activation time is taken out of the event queue, the simulation time is set to this value, and the event is executed. This essentially presumes a single thread of execution on a single processor.

On the other hand, because of statistical considerations, it is common to split a large simulation job into a batch of independent *replications*, where each replication is started from a different random number seed, everything else being the same. This is done to eliminate the dependency on a single sequence of pseudo-random numbers and to establish confidence intervals on the performance measures. This also gives a natural granularity for a coarse-grained parallel or distributed computation by distributing replications to each processor, and designating one processor to collect the statistics emanating from the computation.

This is exactly what our client/server simulation architecture shown in figure A-1 does. It can assist in the two phases of a typical study; the screening phase where a large number of parameter settings has to be evaluated, and the refinement phase where multiple replications are done of the best parameter settings to draw final conclusions. The architecture has been used successfully on a network of about one dozen Sun Sparcstations, mainly SparcStation 10s. The computers used were physically located in the U.S. and Europe, communicating

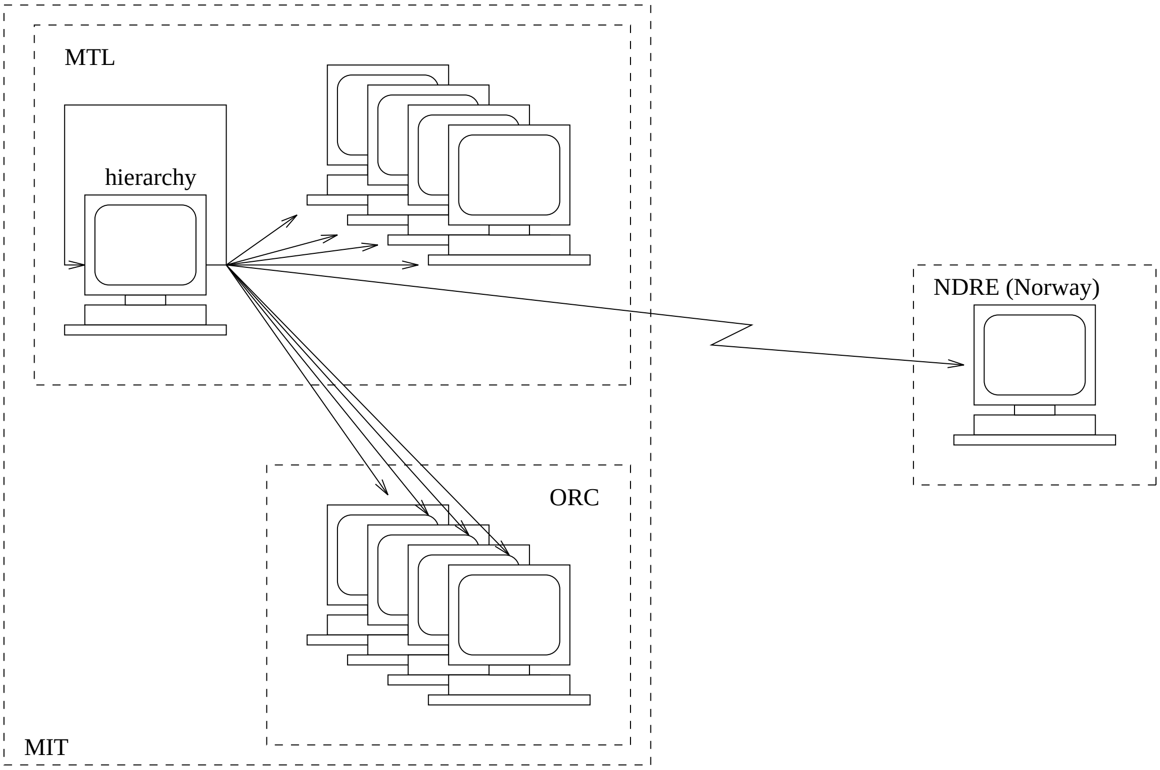

Figure A-1: The architecture used in our simulation experiments

across the Internet. During the study, a Pentium 100MHz PC running Linux was added to the architecture. This demonstrates that the architecture is not Sun-specific. (Incidentally, the PC turned out to work some 20% faster than even a SparcStation 20, possibly because of faster I/O leading to less overhead when starting a client.)

The software has three components: the discrete event simulation program itself, a client program that drives a single replication on some host, and the server program that organizes the computation and collects statistics.

The discrete event simulation is a relatively small and efficient C program. It occupies about 140K of memory on a Sun Sparcstation (which is smaller than a single `csh` process), and does very little input/output.

The client and server applications are written in Perl. The server starts up a replication on each host (including the one it is executing on, which we will call the *master host*) by running a `rsh` *hostname* `client &` with standard input and standard output redirected to local files on the master host.[1] The client and simulation are executed with priority 10, which minimizes interference with interactive users and other jobs. The server stores the process id number of the `rsh` process in an associative array indexed by hostname. In addition to simulation control parameters, the command line arguments passed to the client

[1]To use `rsh` in this way, it is necessary to have an account on each host computer, and in the home directory of that account have a file `.rhosts` that allows the user to log in from the master host. See `man rsh` for details.

include the name of the master host and a port number.

After starting one job on each host, the server listens for completion notices on a socket connected to this port. The completion notice is written to this socket by the client when the simulation is done, and consists of the client host name and a status word. After that, the client exits, and the `rsh` process also terminates. The server will then know that the client on the named host has completed (successfully or otherwise) and will parse the resulting output to update overall statistics. It then launches another client to the host as needed. When all replications are done, the server calculates and prints the statistics generated by the run.

The other main functionality in the client/server interaction is signal handling. If the server is killed (for instance if the user hits Control-C), this signal will be passed on to the client, and appropriate house cleaning is done before termination.

The server comes in two flavors, `pgstry` and `psgrun`. The first one is for running a single replication of each of a number of input files. A typical invocation is `pgstry *.inp`. The server launches a job corresponding to an input file on each host, and channels the results to a file where the extension `.inp` is replaced by `.gso`. When one job finishes, another job is directed to that host. Not much effort has gone into error handling and recovery for this version, so if some host crashes, no attempt is made to restart that job on some other host. Nor has there been any effort to detect host workload and speed to optimize the job allocation over hosts.

The other version runs multiple replications of a single input file, invoked as `pgsrun -N 150 myfile.inp`. This program will run until the number of completion notices asked for (150 in this case) has been received. The replications use random number seeds 1, 2, ... up to the number of replications.

Both server versions have some degree of automatic load balancing between hosts, since job allocations are made when earlier jobs are completed. This will lead to a slow host receiving fewer jobs than a fast one, and if the apparent speed changes due to workload fluctuations during a run, the workload allocation will shift to reflect this.

There is a tradeoff between using few and long simulations versus many short ones. The load balancing tends to work better with many jobs, since there is less chance of a "slow" host holding up the rest of the computation. On the other hand, a new client and simulation program needs to be launched for each new job, so the administrative overhead increases with the number of jobs. Each individual job must also be sufficiently long to reach steady state conditions. A tradeoff was made by trial and error.

By using these programs, we successfully simulated several thousand different cases. Most cases were run for a single replication of 100 weeks of simulated time, but a few hundred cases were selected for further scrutiny. This required more replications of each case. We estimated this to take about a year of CPU time on a single SparcStation 2. Using our virtual supercomputer, the entire job was done in a few days of real time, accumulating some 100,000 years of experience in operating the systems under study.

More speed is available by using more hosts. The approach is not limited to Sun computers, as our Pentium PC demonstrated. Anything that supports UNIX stream sockets and BSD `rsh` can be used. This allows other fast computers to be added by the simple expedient of getting an account, recompiling the simulation for that machine type, and adding the host name to the server list of target hosts in the server script.

In conclusion, our approach gives supercomputer speed at very little cost by using otherwise idle CPU cycles on ordinary UNIX workstations.

Bibliography

Alvarez Vargas, R., Y. Dallery, and R. David (1991a). Une méthode analytique pour l'évaluation des performances d'une ligne de fabrication dont les machines ont des vitesses différentes. Note Interne 91/14, Laboratoire d'Automatique de Grenoble.

Alvarez Vargas, R., Y. Dallery, and R. David (1991b). A study of the continuous flow model of production lines with unreliable machines and finite buffers. In *IMACS-IFAC Symposium on Modelling and Control of Technological Systems*, Lille, France, pp. 221–234.

Arnell, A. (1963). *Standard graphical symbols: A comprehensive guide for use in industry, engineering, and science.* New York: McGraw-Hill.

Arnold, S. F. (1990). *Mathematical Statistics.* Englewood Cliffs, New Jersey: Prentice Hall.

Bai, S. X. (1991). *Scheduling manufacturing systems with work-in-process inventory control.* Ph. D. thesis, Massachusetts Institute of Technology.

Berkley, B. J. (1991). Tandem queues and kanban-controlled lines. *International Journal of Production Research 29*(10), 2057–2081.

Berkley, B. J. (1992). A review of the kanban production control research literature. *Production and Operations Management 1*(4), 393–411.

Bertsimas, D. and G. Mourtzinou (1994). Queues: Theory and applications. Lecture notes for MIT course 6.264J, fall 1994.

Bielecki, T. and P. R. Kumar (1988). Optimality of zero-inventory policies for unreliable manufacturing systems. *Operations Research 36*(4), 532–541.

Bitran, G. R. and L. Chang (1987). A mathematical programming approach to a deterministic kanban system. *Management Science 33*(4), 427–441.

Bonvik, A. M. (1994). Introduction to the CAFE scheduling and simulation testbed. CIDM Memo 94-11, Massachusetts Institute of Technology.

Bonvik, A. M. (1996). Partial differential equations and boundary conditions for a two-machine line in hybrid control. Working Paper LMP-96-002, MIT Laboratory for Manufacturing and Productivity.

Burman, M. H. (1995). *New results in flow line analysis.* Ph. D. thesis, Massachusetts Institute of Technology. Also available as Report LMP-95-007, MIT Laboratory for Manufacturing and Productivity.

Buzacott, J. A. and L. E. Hanifin (1978). Models of automatic transfer lines with inventory banks — a review and comparison. *AIIE Transactions 10*(2), 197–207.

Buzacott, J. A. and J. G. Shantikumar (1992). A general approach for coordinating production in multiple-cell manufacturing systems. *Production and Operations Management 1*(1), 34–52.

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

Carrascosa, M. (1995). Variance of the output in a deterministic two-machine line. Master's thesis, Massachusetts Institute of Technology. Also available as Laboratory for Manufacturing and Productivity report LMP-95-010.

Choong, Y. F. and S. B. Gershwin (1987). A decomposition method for the approximate evaluation of capacitated transfer lines with unreliable machines and random processing times. *IIE Transactions 19*(2), 150–159.

Clark, A. J. and H. Scarf (1960). Optimal policies for the multi-echelon inventory problem. *Management Science 6*(4), 475–490.

Conway, R., W. Maxwell, J. O. McClain, and L. J. Thomas (1988). The role of work-in-progress inventory in serial production lines. *Operations Research 36*(2), 229–241.

Dallery, Y. (1994). On modeling failure and repair times in stochastic models of manufacturing systems using generalized exponential distributions. *Queuing Systems 15*, 199–209.

Dallery, Y., R. David, and X.-L. Xie (1988). An efficient algorithm for analysis of transfer lines with unreliable machines and finite buffers. *IIE Transactions 20*(3), 280–283.

Dallery, Y., R. David, and X.-L. Xie (1989). Approximate analysis of transfer lines with unreliable machines and finite buffers. *IEEE Transactions on automatic control 34*(9), 943–953.

Dallery, Y. and S. B. Gershwin (1992). Manufacturing flow line systems: A review of models and analytical results. *Queuing Systems Theory and Applications 12*(1-2), 3–94. Special issue on queuing models of manufacturing systems.

Dallery, Y. and G. Liberopoulos (1995). Extended kanban control system: A new kanban-type pull control mechanism for multi-stage manufacturing systems. Unpublished manuscript.

Davis, W. J. and S. J. Stubitz (1987). Configuring a kanban system using a discrete optimization of multiple stochastic responses. *International Journal of Production Research 25*(5), 721–740.

de Koster, M. B. M. (1988). *Capacity oriented analysis and design of production systems*. Ph. D. thesis, Technische Universiteit Eindhoven.

Di Mascolo, M., R. David, and Y. Dallery (1991). Modeling and analysis of assembly systems with unreliable machines and finite buffers. *IIE Transactions 23*(4), 315–330.

Di Mascolo, M., Y. Frein, and Y. Dallery (1996). An analytical method for performance evaluation of kanban controlled production systems. *Operations Research 44*(1), 50–64.

Drake, A. W. (1967). *Fundamentals of applied probability theory.* McGraw-Hill.

Feigin, G., J. Fowler, J. Robinson, and R. Leachman (1994). Semiconductor wafer manufacturing data format specification. Technical report, Sematech.

Frein, Y., C. Commault, and Y. Dallery (1994). Modeling and analysis of closed-loop production lines with unreliable machines and finite buffers. Technical Report LAG No. 92-123, revised Nov 1994, Laboratoire d'Automatique de Grenoble, URA 228 CNRS, ENSIEG, BP 46, 38402 Saint Martin d'Hères. To appear in IIE Transactions.

Frein, Y., M. Di Mascolo, and Y. Dallery (1994). On the design of generalized kanban control systems. To appear in *International Journal of Operations and Production Management*, special issue on Modelling and Analysis of Just-in-Time Manufacturing Systems.

Fry, T. D., J. F. Cox, and J. H. Blackstone, Jr. (1992). An analysis and discussion of the optimized production technology software and its use. *Production and Operations Management 1*(2), 229–242.

Gerald, C. F. and P. O. Wheatley (1994). *Applied numerical analysis* (5th ed.). Reading, Mass.: Addison-Wesley.

Gershwin, S. B. (1987a). An efficient decomposition method for the approximate evaluation of tandem queues with finite storage space and blocking. *Operations Research 35*, 291–305.

Gershwin, S. B. (1987b). Representation analysis of transfer lines with machines that have different processing rates. *Annals of Operations Research 9*, 511–530.

Gershwin, S. B. (1989). An efficient decomposition algorithm for unreliable tandem queuing systems with finite buffers. In H. G. Perrios and T. Altiok (Eds.), *Queuing networks with blocking*, pp. 127–146. Amsterdam: North-Holland.

Gershwin, S. B. (1991). Assembly/disassembly systems: An efficient decomposition algorithm for tree-structured networks. *IIE Transactions 23*(4), 302–314.

Gershwin, S. B. (1994). *Manufacturing Systems Engineering.* Englewood Cliffs, New Jersey: PTR Prentice Hall.

Gershwin, S. B. and O. Berman (1981). Analysis of transfer lines consisting of two unreliable machines with random processing times and finite storage buffers. *AIIE Transactions 13*(1), 2–11.

Glassey, C. R. and Y. Hong (1986). The analysis of behavior of an unreliable n-stage automatic transfer line with $(n-1)$ inter-stage buffer storages. Technical report, Dept. of Industrial Engineering, University of California, Berkeley.

Goldratt, E. and J. Fox (1984). *The Goal.* North River Press.

Karmarkar, U. S. (1986). Kanban systems. Working Paper QM8612, Graduate School of Management, University of Rochester.

Kimball, G. (1988). General principles of inventory control. *Journal of Manufacturing and Operations Management 1*(1), 119–130.

Kimemia, J. and S. B. Gershwin (1983). An algorithm for the computer control of a flexible manufacturing system. *IIE Transactions 15*(4), 353–362.

Kimura, O. and H. Terada (1981). Design and analysis of pull system, a method of multi-stage production control. *International Journal of Production Research 19*(3), 241–253.

Kleinrock, L. (1975). *Queuing Systems, Volume I: Theory.* New York: John Wiley & Sons.

Le Bihan, H. (1994). Méthodes approximatives d'étude des lignes de production. Dea systèmes informatiques, Université Pierre et Marie Curie.

Lee, Y.-J. and P. Zipkin (1992). Tandem queues with planned inventories. *Operations Research 40*(5), 936–947.

Little, J. D. C. (1961). A proof of the queueing formula $L = \lambda W$. *Operations Research 9*, 383–387.

Liu, X.-G. and J. A. Buzacott (1990). Approximate models of assembly systems with finite inventory banks. *European Journal of Operations Research 45*, 143–154.

McIlrath, M. B., D. E. Troxel, D. S. Boning, M. L. Heytens, P. Penfield, Jr., and R. Jayavant (1990). CAFE — the MIT computer aided fabrication environment. In *Proceedings 1990 IEMT symposium*, pp. 297–305. IEEE/CHMT.

Mitra, D. and I. Mitrani (1990). Analysis of a kanban discipline for cell coordination in production lines, I. *Management Science 36*(12), 1548–1566.

Mitra, D. and I. Mitrani (1991). Analysis of a kanban discipline for cell coordination in production lines, II: Stochastic demands. *Operations Research 39*(5), 807–823.

Monden, Y. (1983). *Toyota Production System: Practical approach to production management.* Atlanta: Industrial Engineering and Management Press.

Ohno, K., K. Nakashima, and M. Kojima (1995). Optimal numbers of two kinds of kanbans in a jit production system. *International Journal of Production Research 33*(5), 1387–1401.

Powell, S. G. and D. F. Pyke (1994). An empirical investigation of the two-moment approximation for production lines. *International Journal of Production Research 32*(5), 1137–1157.

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo method.* New York: John Wiley and Sons.

Rubio, R. and L. M. Wein (1994). Setting base stock levels using product-form queuing networks. Submitted for publication.

Schor, J. E. (1995). Efficient algorithms for buffer allocation. Master's thesis, Massachusetts Institute of Technology. Also available as MIT Laboratory for Manufacturing and Productivity report LMP-95-006.

Sedgewick, R. (1992). *Algorithms in C++.* Addison-Wesley.

Sematech (1995). Semiconductor testbed datasets. Available by anonymous ftp from *ftp.sematech.org.*

Semery, A. (1987). Modélisation des LIF: Intégration du modèle de Dubois et Forestier à l'heuristique de Gershwin, pur les lignes à temps de cycle differentes. Rapport de recherche AS/619/87, Regie Renault, Direction de la Recherche.

Senge, P. M. (1990). *The fifth discipline : The art & practice of the learning organization.* Currency Doubleday.

Shingo, S. (1989). *A Study of the Toyota Production System from an Industrial Engineering Viewpoint* (2nd ed.). Productivity Press.

So, K. C. and S. C. Pinault (1988). Allocating buffer storages in a pull system. *International Journal of Production Research 15*(12), 1959–1980.

Spearman, M. L., D. L. Woodruff, and W. J. Hopp (1990). CONWIP: a pull alternative to kanban. *International Journal of Production Research 28*(5), 879–894.

Stewart, D. E. and Z. Leyk (1994). Meschach library, version 1.2b. Available from *http://www.netlib.org/c/meschach* and *ftp://thrain.anu.edu.au/pub/meschach.*

Sugimori, Y., K. Kusunoki, F. Cho, and S. Uchikawa (1977). Toyota production system and kanban system materialization of just-in-time and respect-for-human system. *International Journal of Production Research 15*(6), 553–564.

Tabe, T., R. Muramatsu, and Y. Tanaka (1980). Analysis of production ordering quantities and inventory variations in a multi-stage production ordering system. *International Journal of Production Research 18*(2), 245–257.

Van Ryzin, G. (1987). Control of manufacturing systems with delays. Master's thesis, Massachusetts Institute of Technology. Also available as MIT Laboratory for Information and Decision Systems Report LIDS-TH-1676.

Van Ryzin, G., S. X. C. Lou, and S. B. Gershwin (1993). Production control for a tandem two-machine system. *IIE Transactions 25*(5), 5–20.

Veatch, M. H. and L. M. Wein (1994). Optimal control of a two-station tandem production/inventory system. *Operations Research 42*(2), 337–350.

Wein, L. M. (1988). Scheduling semiconductor wafer fabrication. *IEEE Transactions on Semiconductor Manufacturing 1*(3), 115–130.

Womack, J. P., D. T. Jones, and D. Roos (1990). *The machine that changed the world : The story of lean production.* Rawson Associates.

Yan, H., S. Lou, S. Sethi, A. Gardel, and P. Deosthali (1994). Testing the robustness of various production control policies in semiconductor manufacturing. Forthcoming in *IEEE Transactions on Semiconductor Manufacturing.*