SCHOOL OF OPERATIONS RESEARCH
AND INDUSTRIAL ENGINEERING
COLLEGE OF ENGINEERING
CORNELL UNIVERSITY
ITHACA, NY 14853-3801

TECHNICAL REPORT NO. 1056

May 1993

A COMPARISON OF ALTERNATIVE KANBAN CONTROL MECHANISMS: PART 2 EXPERIMENTAL RESULTS[1]

by

John A. Muckstadt and
Sridhar R. Tayur

[1]This work was supported in part by National Science Foundation Grant DDM-9215328.

A Comparison of Alternative Kanban Control Mechanisms: Part 2 Experimental Results

John A. Muckstadt
School of Operations Research and Industrial Engineering
Cornell University, Ithaca, NY 14850

Sridhar R. Tayur
Graduate School of Industrial Administration
Carnegie Mellon University, Pittsburgh, PA 15213

May 1993

Abstract

This paper continues our study of the design of inventory control policies for serial systems. Using simulation, we compare alternate control policies where a target of mean throughput has to be met under three different objectives – minimize the maximum inventory, minimize the average inventory and minimize the variance of the output – for balanced lines and lines with one bottleneck.

The main objectives of this paper are (1): to study several objectives not amenable to sample path analysis and (2) to show some sets of conditions and objectives under which CONWIP is preferred over the traditional *kanban* mechanism, and vice-versa.

1 Introduction

This paper is a sequel to Muckstadt and Tayur(1993) and studies several objectives of serial lines not amenable to sample path analysis.

In Muckstadt and Tayur(1993) we demonstrated the following.

1. CONWIP and the conventional kanban control are just two extremes in a finite family of implementable *pull* control systems.
2. Two of the structural results of Tayur(1992) –Allocation and Partition– continue to hold even in presence of stochastic demand and raw material processes.
3. There is a difference between yield losses and the sources of variability considered in Tayur(1992) –processing time variability, rework and machine breakdown – when it comes to the reversibility property.

In this paper, we show several sets of conditions and performance criteria under which CONWIP is preferred over the traditional *kanban* mechanism, and vice-versa. This is done via simulation, as many of the performance criteria such as average inventory and variance of the output are beyond the scope of sample path analysis.

We briefly present the model introduced by Mitra and Mitrani(1990) and paraphrase the description from Muckstadt and Tayur(1993).

2 The Model

We study a serial manufacturing system that uses a general *kanban* control mechanism. Processing times are variable, machine breakdowns are possible, rework may be required and yield is not perfect (the yield at any processing step is random). The serial production line we will study consists of M machines arranged in a series (or in tandem). These M machines are partitioned into N *cells*, each consisting of a set of machines grouped together such that the total number of kanbans for this group is fixed. Thus, a cell is simply a kanban loop. A *cell partition* is a collection of non-overlapping and collectively exhaustive groups of consecutive machines. If all the M machines are in the same *cell*, we have a CONWIP (CONstant-Work-In-Process) type control system; if, on the other hand, there are a total of M cells, each cell containing exactly one machine, we have a *traditional kanban control system* (TKCS). To formalize our ideas and to make our exposition precise, we introduce the following (mathematical) description of a control system.

We will use $\mathrm{N}/(M_1,\ldots,M_N)/(C_1,\ldots,C_N)$ to denote a serial production line with N cells, M_i machines in cell i, C_i white kanbans in cell i, $i = 1 \ldots$ N. By *allocation* we mean the vector $(C_1,\ldots,C_N)$, and by *partition* we mean $(M_1,\ldots,M_N)$. The set $\{\ \mathrm{N}/(M_1,\ldots,M_N)/(C_1,\ldots,C_N)$: $\sum_{i=1}^{N} M_i = M, M_i \geq 1, \sum_{i=1}^{N} C_i = C, C_i \geq 1, N \leq M\}$ contains all possible *configurations* for a line with M machines and C white kanbans. Using this notation, we see that CONWIP is $1/(\mathrm{M})/(\mathrm{C})$ system, and TKCS is a $\mathrm{M}/(1,\ldots,1)/(C_1,\ldots,C_M)$ system. All other configurations give rise to other possible designs within this *family* of controls. Henceforth, we will refer to the general control scheme as *kanban* control.

We briefly describe the essentials of a single-product kanban controlled system (Figure 1).

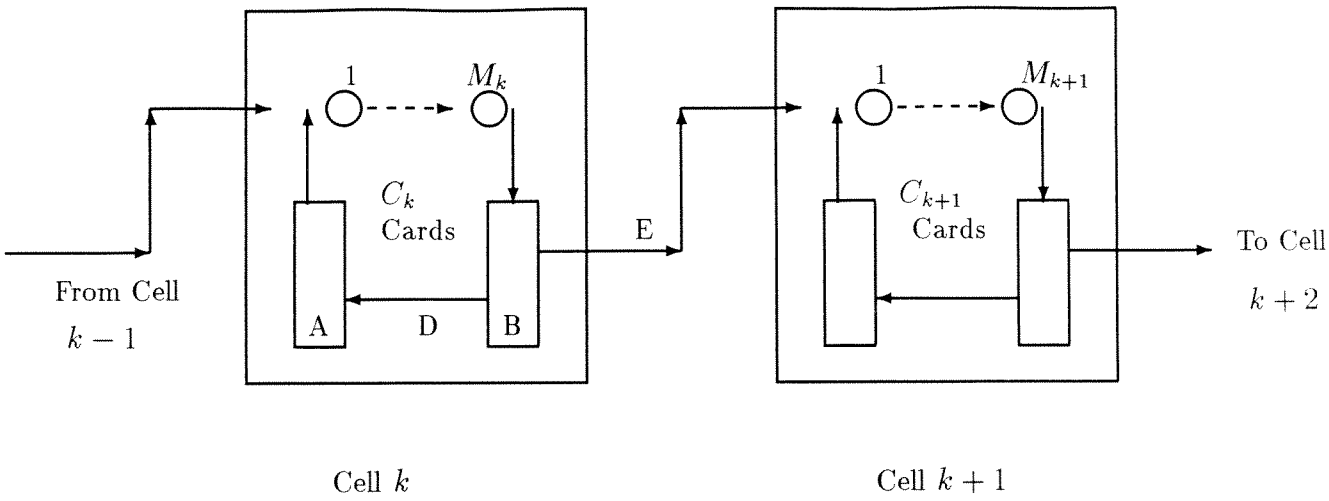
As shown in Figure 1, a cell consists of

1. machines in tandem – the processing times on the machines may be stochastic, and all parts go through each machine exactly once.

2. an output hopper – in which batches of material that have completed all operations in the cell (and have not suffered a complete loss) wait for withdrawal by the successor cell.

3. a bulletin board – where requests are posted for material from the predecessor cell, in the form of kanbans. (These kanbans were assumed to be white in color in Muckstadt and Tayur 1993.)

The product moves through the line in batches, which can be of size one. The service discipline is first-come, first-served, and each machine can process only one part at a time. No preemptions are allowed. The parts completed in cell $k-1$ become the input material for cell k, for k=2,..., N. A batch must acquire one of these cards in order to enter the cell, and must continue to hold it throughout its stay in that cell. After a batch has been completed in cell k, it is placed in the output hopper with its kanban, awaiting admission into the next cell. If there is a *complete* yield loss at a particular machine in a cell (say in



A: Bulletin Board of Cell k

B: Output Hopper of Cell k

D: Card traveling from Output Hopper to Bulletin Board

E: Part transferred to Cell $k+1$ from Cell k

Figure 1: Cell Description.

cell k, *all* items in the batch are scrapped), then the batch is thrown away and the white kanban that was attached to this (rejected) batch is placed on the bulletin board of cell k, signalling a need for replenishment. This *immediate* pull response to a yield loss is an attractive quality of this mechanism. (If at the end of a processing step a batch contains at least one good item, then it is sent to the next processing stage. The determination of the number of *non-defective* items in a batch is made at the end of the processing of the batch.) Both rework and machine breakdowns are accommodated by a suitable change to the form of the processing time distribution at a machine.

Note that the mechanism is *pull* between cells, and *push* within a cell. Also note that it is not possible for both the output hopper of cell k and the bulletin board of cell $k+1$ to be simultaneously non-empty. If a kanban is present on the bulletin board of cell $k+1$, and a batch is available in the output hopper of cell k, the batch would be moved to the queue in front of the first machine in cell $k+1$ along with the kanban from the bulletin board of cell $k+1$. Thus, the maximum inventory possible in cell k is C_k batches, and no inventory can sit between adjacent cells. This is how kanbans control inventory in the cells. When a completed part is withdrawn to the next cell (cell $k+1$) the kanban of cell k stays within the cell, and is posted on the bulletin board of cell k. This is a signal to the preceding cell, cell k-1, that cell k needs a part. Thus, kanbans also serve as an information system that controls material transfer between successive cells.

In our context, then, the problem of buffering a M machine serial production line is

Muckstadt and Tayur(1993)		
Decision	Objective	Method of Study
Allocation	Min. Max. Inventory	Sample Path
Partition	Min. Max. Inventory	Sample Path
Sequencing	Max. Throughput	Sample Path
This Paper		
Allocation	Min. Max. Inventory	Heuristics/Simulation
Allocation	Min. Av. Inventory	Heuristics/Simulation
Partition	Min. Av. Inventory	Simulation
Partition	Min. Coeff of Variation of Output	Simulation
Sequencing	Max. Throughput	Simulation

Figure 2: Summary of Issues and Methods used to study them.

equivalent to partitioning the line into N cells, allocating a certain number of kanbans to each cell. There is no reason, *apriori*, to expect any one control from the above *family* to be superior to all others in all possible scenarios. In particular, neither CONWIP nor the traditional kanban control can claim superiority over another in all situations. However, some controls will be superior to others for particular objectives. We will illustrate this fact subsequently.

The remainder of this paper is organized as follows (Figure 2 provides a brief summary of topics studied here as well as in Muckstadt and Tayur(1993)). In section 3, we discuss the allocation, partition and sequencing decisions when the constraint is on average inventory as well as maximum inventory, where we show that for balanced lines, TKCS may be the best strategy for the former. Recall from Muckstadt and Tayur(1993) that for the latter objective, the optimal partition is CONWIP. We also the study the latter objective – and provide good allocation policies– for a partition other than CONWIP. In section 4, we conclude this paper where some of the practical issues that arise are considered; results from this paper and the previous one are used to answer some common questions. The appendix records details about our simulation experiments.

3 Analysis of Optimal Strategies under Different Objectives

In this section, we consider the objectives of minimizing the average inventory and minimizing the maximum inventory while meeting a target throughput for balanced lines as well as lines with one distinct bottleneck. The design parameters at our disposal are the partition and the allocation of white cards. This study is done primarily using simulation; however, the structural results of the previous paper (Muckstadt and Tayur 1993) are freely used in reducing the simulation effort. First, we need the following definitions.

A line with M machines is *balanced* if:

1. the yield loss distribution is the same on each machine.

2. the coefficient of variation (ratio of the standard deviation to the mean) of the processing times is the same across the machines, and

3. the traffic intensity (ratio of the arrival rate to the service rate) at each machine is the same.

Other somewhat equivalent defintions for balanced lines have been suggested; see Conway et. al (1988) for one.

A *bottleneck* is a machine that has a higher mean processing time than the others with the same coefficient of variation, or a higher coefficient of variation than the others if the mean is the same, or both. It is possible to have pairs of machines that cannot be ordered in the above manner–one machine in the pair has a higher mean processing time but a lower variance than the other. This case is usually harder to analyze analytically as the particular distributions of the processing times are important to determine the "slower" machine in buffered lines. A simple case of this type is analyzed in Tayur(1990); we do not consider such cases here. Thus, in our discussions in this paper, a bottleneck machine has at least as high a mean and variance as the rest of the machines.

A machine that is closer to the end of the processing sequence than another is said to be *downstream* of the other. We define *upstream* analogously.

To our knowledge, it is not analytically tractable to study cases of fixed average inventory. Hence, we have conducted our analysis using simulation. The results of our experiments are displayed in a series of graphs. All the graphs show points in the sets of *efficient frontiers*. As each curve in a graph is a plot of *average* inventory versus mean throughput, the lower the curve, the more desirable is the control policy that generates it. Recall that the control policy is determined by the *partition*. In a 3 machine line we have four options for control policies: 1/(3)/() or CONWIP, 2/(1,2)/(), 2/(2,1)/() and 3/(1,1,1)/() or TKCS. When the partition is other than CONWIP, it is necessary to distinguish the points (on the same curve) as there are several different *allocation* rules that may be used. Thus, for TKCS, we study among the many possible, three different allocation rules; these are described subsequently. The points are distinguished in the graph by using $\Box$, $\diamond$, $\circ$ and other such markers. As the number of allocations plotted is large, and in many cases the different allocation rules perform equally well, the curves tend to be clustered; to avoid further confusion, we have refrained from connecting these points by lines. An efficient frontier, then, is the lower envelope of the points generated by the combination of an allocation rule and a partition.

Unless otherwise mentioned, the mean processing time on the machines is 0.1 (equivalently, a rate of 10 parts/ unit time). The details of the simulation experiment is provided in Remark 7 of the appendix.

The decisions to be made while designing a serial line can be divided into the following categories:

1. Given a set of operations that need to be performed on every part, in what order should they be done (if there is some flexibility) ?. This is the *sequencing* issue.

2. Given a *sequence* of machines, is there any benefit of grouping some machines together from an operational point of view? This is the *partitioning* issue.

3. Given a *partition* of the line and a target throughput, how many total kanbans are required ?, and how many should be placed in each cell? This is the *allocation* issue.

The answers to the above questions will be provided from the *bottom-up*, that is, we first resolve the *allocation* problem, then the *partition* question and lastly the *sequencing* issue.

3.1 Allocation

Given a $M/(1,\ldots,1)/()$ partition, and a target throughput rate, we need to make two (simultaneous) decisions: what is the required total number of cards, and how should these cards be allocated. We first give a brief summary of two heuristics that will be used in making the above decisions.

3.1.1 Heuristics for balanced lines

We look for heuristic methods because exact computation of throughput for a given allocation and partition, even for an environment with all machines having an exponential distribution, seems computationally intractable for large problems. For machines that have a non-exponential processing time distribution, an exact analytic solution may not be possible even for small problems.

A practical method of arriving at a good allocation is in two steps: (1) for every candidate allocation calculate its measure of goodness (or effectiveness), and (2) pick the allocation with the highest measure. In the past, an estimate of the mean throughput obtained from analytical approximations was used as a measure of goodness. This severely restricts the range of processing times that can be analyzed, and takes significant computing time and space. We recognize that the major stumbling block in previous approaches is in the time needed to estimate the mean throughput given a particular allocation.

After simulating a number of small lines (four-cell lines with one machine per cell; see Tayur(1992) and Tables 1-2 below), we observed that the optimal allocation of C cards to a balanced line was insensitive to the processing time distribution or the yield loss distribution on the machines. As an example, consider allocating 10 cards to a four-cell line with one machine per cell. As long as the four machines are chosen so that the line is balanced, the optimal allocation is (1,4,4,1). The problem of allocating cards to cells in a balanced line appears to be a combinatorial issue rather than an issue in stochastic processes. Of course, the throughput achieved in the different cases is not the same. In the tables below, **A** stands for a line with yield loss probability of 0.25, while **B** stands for yield loss of 0.5. These results supplement the tables in Tayur(1992) where there were no yield losses.

Using the above observation, we avoid the problem of not being able to compute the mean throughput by developing surrogate measures for mean throughput. These measures do not depend on the distributions of the processing times (because the line is balanced),

Table 1: State-Space and Pulse in a 4 cell line with exponential machines. Throughput is measured in number of units produced per unit of time.

allocation	states	pulse	Throughput A	Throughput B
4 machines				
1,3,2,1	81	0.6944	2.119	0.4160
1,4,1,1	72	0.6701	1.989	0.3890
1,3,3,1	115	0.7261	2.236	0.4418
1,4,2,1	110	0.7182	2.190	0.4305
1,4,4,1	204	0.7696	2.388	0.4729
1,5,3,1	198	0.7652	2.359	0.4660
1,6,5,1	405	0.8130	2.533	0.5031
1,7,4,1	390	0.8083	2.507	0.4968

Table 2: Throughput in four 4/(1, ...,1)/() lines: with erlang(2) machines and yield loss of 0.25 at each station, with erlang(2) machines and yield loss of 0.5 at each station, with deterministic processing times and yield loss of 0.25 at each station and with deterministic processing times and yield loss of 0.5 at each station. Throughput is measured in number of units produced per unit of time.

allocation	Throughput A	Throughput B	Throughput A	Throughput B
4 machines	erlang(2)	erlang(2)	deter.	deter.
1,3,2,1	2.235	0.4504	2.813	0.5166
1,4,1,1	2.189	0.4204	2.643	0.4857
1,3,3,1	2.460	0.4743	2.887	0.5365
1,4,2,1	2.415	0.4650	2.861	0.5290
1,4,4,1	2.600	0.5031	2.961	0.5585
1,5,3,1	2.577	0.4986	2.951	0.5549
1,6,5,1	2.725	0.5308	3.020	0.5768
1,7,4,1	2.705	0.5264	3.013	0.5740

but on the number of cells and the number of kanbans in each cell. The heuristics, then, *maximize* the surrogate measures. Our measures are combinatorial in nature, and can be computed very efficiently by recursive methods (Tayur(1992)).

The first of the two measures is called *state-space*; the other is *pulse*. The following example is useful for motivation of the state-space measure.

Example 1 *Consider a two machine line with identical exponential machines. Let the total number of kanbans be C, with C_1 in cell 1, and C_2 in cell 2. This can easily be modeled as a birth-death process (a special case of Markov chains) with a total of C+1 states. Each state represents the difference between the number of cards in the output-hopper of cell 1 and on the bulletin board of the second cell, and can take values in the set $\{C_1, \ldots, +1, 0, -1, \ldots, -C_2\}$. Each state is equally likely. In state C_1, machine 1 is idle; in all other states it is processing a part. Adding an extra card to either cell has exactly the same effect on the size of the state-space and the throughput: The size of the state-space increases by one, and the throughput of the line increases as the probability of being in state C_1 decreases. Compare this to Corollary 6 of Muckstadt and Tayur (1993a).*

Hence all we need do is *count* the total number of states in the Markov chain that results due to an allocation to a line with all the machines having an exponential processing time distribution. Given a $N/(M_1, \ldots, M_N)/()$ line with a total of C cards to be allocated, the heuristic is to pick the allocation that creates the *largest* markov chain. The recursion for the computation of the number of states given an allocation has complexity $\mathcal{O}(N)$ for a $N/(1, \ldots, 1)/()$ line and $\mathcal{O}(NM^2C^2)$ for a $N/(M_1 \ldots M_N)/()$ line. We could use a different recursion in case the processing time distributions were Erlang, but this is not required because they lead to the same allocation (a consequence of the Schur-concavity property of the measure (see Tayur(1992)). A second example emphasizes this approach; compare the result with Corollary 5 of Theorem 1 of Muckstadt and Tayur (1993a).

Example 2 *For a 3/(1,1,1)/(1,1,1) line with all machines having exponential processing time distributions, the number of states in the Markov chain is 8. Placing a second card in the first cell leads to the allocation (2,1,1) which creates a Markov chain with 11 states, while an allocation (1,2,1) creates a Markov chain with 13 states. Consequently, we choose allocation (1,2,1) over allocation (2,1,1).*

The second measure, which we call *pulse*, is illustrated by Example 3. The pulse assumes that each state in the Markov chain is equally likely. The pulse of a system equals the *average* transition rate out of the states. If the underlying Markov transition probability matrix were doubly stochastic, the pulse would equal the mean throughput. The computation is done by first summing (across the states) the number of machines working in any state, and then this sum is divided by the total number of states and the total number of machines in the line. Intuitively, we are computing the fraction of time a particular machine is working assuming that all the states of the Markov chain are equally likely.

Example 3 *Consider a two-machine two cell line with identical exponential machines with means 1. Then pulse* $= \frac{1+2(C-1)+1}{2(C+1)} = \frac{C}{C+1}$ *=mean throughput rate.*

As in the case of the state-space measure, the recursion for the computation of pulse for a given allocation has complexity $\mathcal{O}(N)$ for a $N/(1,\ldots,1)/()$ line and $\mathcal{O}(NM^2C^2)$ for a $N/(M_1,\ldots,M_N)/()$ line.

Our computations show that the state-space measure works well when the ratio of number of cards to the number of cells is low (≤ 1.5), while the pulse works well when the ratio of number of cards to the number of cells is high (≥ 2.5). In the range 1.5 - 2.5, both the measures coincide in picking the allocation. The intuitive explanation for this observation is given in Tayur(1990).

3.1.2 Performance of Heuristics under different objectives

Recall that we are interested in meeting a target throughput rate in an efficient manner. Two objectives that we study in detail are: (1) minimize the *maximum* inventory, and (2) minimize the *average* inventory. As we shall soon show, the two lead to different allocations. Note that minimizing the average inventory is equivalent to minimizing flow time. It turns out that the allocation that provides the minimum average inventory also provides the lowest co-efficient of variation of the departure process out of the line.

Procedure 1: If our objective is to minimize the *maximum* inventory, we would want to achieve the desired throughput with a *minimum* number of kanbans. Let C be the total number of kanbans needed. First, the end cells would be allocated exactly one card each. If M =2, for each C (starting from 2) we would simulate a 2/(1,1)/(1,C-1) line until the target throughput is met. If M=3, we would simulate for each C (starting from 3) a 3/(1,1,1)/(1,C-2,1) line until the target throughput is reached. This follows from Theorem 1 of Muckstadt and Tayur(1993a). If M $\geq$4, then we need to simulate more than one allocation for each C as no theoretically supported heuristic has been obtained thus far that can reduce the simulation effort to one per every value of C for *non-balanced* lines.

If, however, we have a *balanced* line, then both the *state-space* and *pulse* heuristics are used to determine the *allocation* as follows. For every fixed C, the heuristics provide the allocation that needs to be simulated. Thus, starting from a value of M, we can increase the value of C, one at a time; and for each value of C, instead of simulating a large number of feasible allocations, we simulate only the one that the heuristics suggest. As an example, in a 4 cell line with 10 cards to be allocated, the *allocation* picked by the heuristics is (1,4,4,1). In a 4 cell line, it so happens that the best allocation (for every C) has the property that the difference between the number of cards in the two interior cells is at most 1. This happens because there is more benefit in buffering a balanced line uniformly, rather than concentrating more on a portion of the line and neglecting the other (equally variable) portion.

Procedure 2: If our objective is to minimize the *average* inventory, the procedure to determine the allocation is different. Consider the case of a $M/(1,\ldots,1)/()$ balanced line. We

begin with a total of M cards, one in each cell, and gradually increase the total number of cards until the target throughput is met. As before, each of the end cells needs only one card; but, the additional cards are allocated to the interior cells in a different way from that explained above. The cards are distributed from downstream first, and slowly moved upstream. The reason for this is because a card downstream accounts for a lower average inventory than one upstream (Conway et al.(1988), Mitra and Mitrani(1990)). This phenomenon occurs because upstream the chances of being blocked are higher (leading to cards holding parts in the output hopper), while downstream the chances of starving are higher (meaning that the cards are posted on the bulletin board). To give an example of the allocation procedure in a 4/(1,1,1,1)/() line, we would start with 4 cards and the allocation is (1,1,1,1) by default, then move to 5 cards where we select the allocation (1,1,2,1), and then go to 6 cards. With 6 cards there are two options, namely, (1,1,3,1) and (1,2,2,1). Observe that in procedure 1, the allocation (1,2,2,1) would have been selected by the heuristics. In this procedure, however, allocation (1,1,3,1) is chosen if the ratio of the increase in throughput to the increase in average inventory is higher than that resulting from (1,2,2,1). For a total of seven cards we compare (1,1,4,1) with (1,2,3,1), assuming (1,1,3,1) was chosen in the previous stage. We would continue in this manner until we find a C (and the allocation associated with it) that meets the target throughput. This is clearly a more tedious procedure and yields only a slightly better frontier in a graph of average inventory versus average throughput than does procedure 1 (see Figures 3, 5-12). More substantial benefits can be obtained by selecting the proper *partition*. This is discussed in the next subsection.

A third procedure that was compared is one where the end cell is allowed to have more than one card in it although it may have only one machine.

Figure 3: *The results of the above procedures for a 4/(1,1,1,1)/() line with (identical) exponential service times are shown in the lower curves in Figure 3.* $\Box$ *is the procedure to acheive the target throughput with minimum number of cards (procedure 1),* $\diamond$ *is the second procedure, and* $*$ *is an allocation that has more than one card in the fourth cell. Recall that the lower the curve, the more desirable is the procedure. Note that TKCS has 12-18% less average inventory than CONWIP at equal average throughput values, which is also plotted using* $+$ *on the same graph. Notice that while the points generated for TKCS by the different procedures are clustered closely, the graph of CONWIP differs significantly. Thus, we conjecture that the benefits in refining the* allocation *given by the pulse or the state-space heuristics (once the TKCS partition has been selected) is not as high as selecting TKCS instead of CONWIP.*

The allocation of cards to cells in the N/$(M_1, \ldots, M_N)$/$(C_1, \ldots, C_N)$ case is analogous to that in the case of $N/(1, \ldots, 1)/()$. The difference between the average inventory due to allocations achieved by the different procedures turns out to be small even in this setting.

We remark that in reality, as no line is perfectly balanced, we need to verify that the pulse and state-space measures are robust. Our experimentation indicates that the above

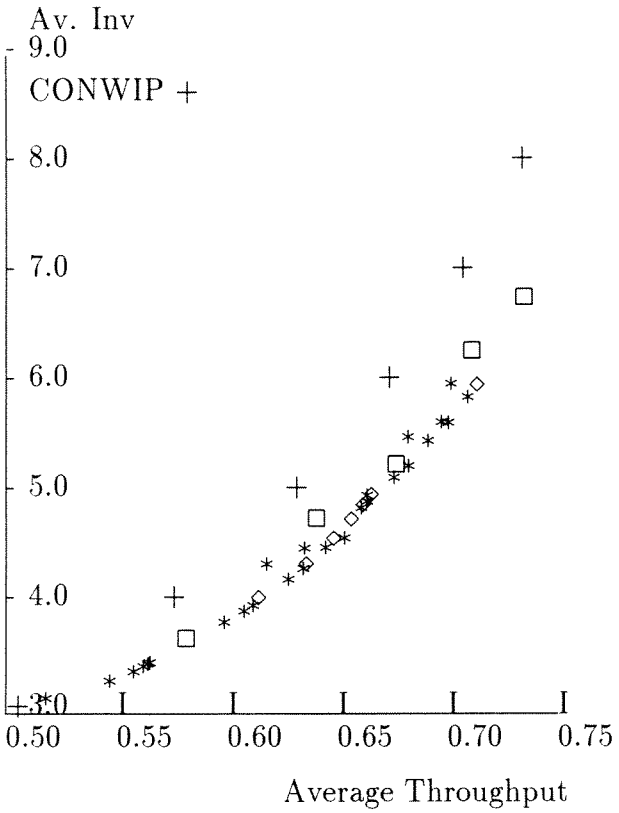


Figure 3: Average inventory vs. mean throughput for a 4 machine line with identical exponential machines. All machines have a mean processing time of 0.1.

heuristics work well (pick the optimal allocation) in the cases where the difference between the bottleneck machines and the other machines is less than 5%. A table is provided in the appendix (remark 6) to demonstrate this.

In summary, given a fixed partition, for the objective of minimizing the average inventory for a particular target throughput, procedure 2 does not perform significantly better than procedure 1 to warrant the increased computation. Procedure 1 is near optimal for the objective of minimizing maximum inventory, as shown in Tayur(1992). Thus, the allocation procedure, for both objectives, can be summarized as follows: For every value of C, the total number of cards in the system, identify the *allocation* that dominates all others; if this is not possible use the state-space measure or the pulse measure to pick one; simulate the line for this allocation. Repeat this procedure until the value of C is found that meets the target throughput.

3.2 Partition

Given a sequence of machines, we are interested in grouping consecutive machines to achieve the *efficient* frontier. If our objective is to minimize the the *maximum* inventory in the system, we have shown in Theorem 6 of Muckstadt and Tayur(1993) that all the machines should be grouped together in one cell.

Figure 4: *We demonstrate the* efficient frontiers *for all the four partitions in a three-machine line in Figure 4. The four possible partitions give rise to 1/3/(.) (+), 2/(2,1)/(.) (◇), 2/(1,2)/(.) (∗) and 3/(1,1,1)/(.) (□) lines. The lower the curve, the more desirable is the partition. Note the* uniform *superiority of TKCS over CONWIP when looking at* average *inventory as the objective; at equal average throughputs, the average inventory of TKCS is lower by over 25%. This is in strong contrast to Theorem 6 of Muckstadt and Tayur (1993a), where TKCS is* uniformly *inferior to CONWIP when the objective is to minimize the* maximum *inventory. Also note that the other two partitions are sandwiched between CONWIP and TKCS.*

The observations made in the above graph are *not* limited to a three-machine line, nor for that matter to a *balanced* line. Recall that we had noticed a similar relationship between CONWIP and TKCS in Figure 3 for a four-machine *balanced* line.

Next, we examine an unbalanced line. We are interested in determining good partitions for lines with a distinct bottleneck. Figures 5-12, show the efficient frontiers for 4 machine lines that contain a bottleneck machine at different locations. Specifically, we attempt to understand how the *severity* and the *location* of the bottleneck in the sequence affect the relative performance of the control strategies. Recall that a bottleneck is a machine that has a higher mean and/or variance in processing time compared to the rest of the machines. To study these aspects, we located the bottleneck at the four positions (one at a time, of

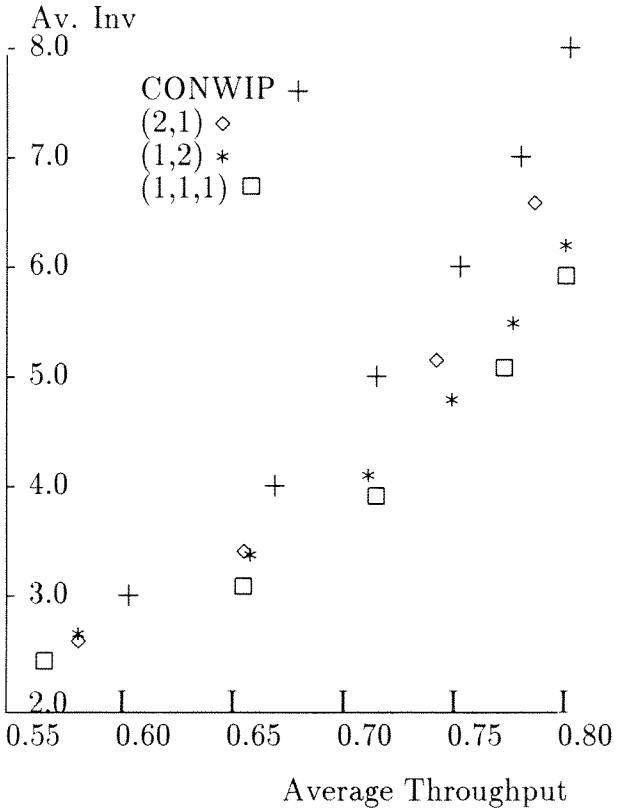


Figure 4: Average inventory vs. mean throughput for a 3 machine line with exponential machines. All machines have a mean of 0.1.

course), and considered five different levels of severity of bottlenecks. We show only a subset of the results. In all the graphs that follow, we use the same markers for TKCS as those used in Figure 3, namely: $\Box$ is the procedure to acheive the target throughput with a minimum number of cards (procedure 1), $\diamond$ is the second procedure, $*$ is an allocation that has more than one card in the fourth cell. Recall that $+$ is used for CONWIP.

Figures 5 and 6: *The effects due to severity of the bottleneck are shown in Figures 5 and 6 for a four machine line. Only the TKCS and the CONWIP configurations are plotted. The markings used in this plot are as those in Figure 3; all three procedures for TKCS are plotted. In all cases, the bottleneck is in the first location. In Figure 5 the mean processing time of the bottleneck is 20% higher than the rest and in Figure 6 it is 50% higher than the rest. To clarify, in Figure 6, for example, the bottleneck has a mean of 0.15, while the other machines have a mean of 0.1 and all the distributions are exponential. In Figure 5, the average inventory for TKCS is 15-30% lower than CONWIP at equal average throughputs; in Figure 6, it is 25-32% lower. Note that as the the bottleneck becomes more severe, the relative performance of TKCS over CONWIP improves. This is because the cards in the downstream cells in TKCS are more likely to be posted on their respective bulletin boards as the severity of the bottleneck increases.*

Figures 7 and 8: *The effect of variance is studied in Figures 7 and 8 for a four machine line. Only the TKCS and the CONWIP configurations are plotted. The bottleneck machine is exponential, while the rest of the line is Erlang(2) in Figure 7, and Erlang(3) in Figure 8. Note that Erlang(k) represents a processing time distribution that is Erlang with parameter k. All the machines have the same mean processing times, namely, 0.1. In Figures 7-8, TKCS has about 10-12% lower average inventory than CONWIP at equal average throughputs. Note that as the bottleneck becomes more severe, the relative performance of TKCS over CONWIP improves. This is because, as before, the cards in the cells downstream of the bottleneck are more likely to be posted on the their respective bulletin boards.*

Figures 9-11: *The effect of the location of the bottleneck can be understood by comparing the curves in Figure 3 (no bottleneck) and Figures 9-11. Only the TKCS and the CONWIP configurations are plotted. The bottleneck has a 20% greater mean processing time compared to the other machines but the coefficient of variation on all the machines are equal. In Figure 9, there is about 10% benefit of TKCS over CONWIP; in Figure 10, it is about 5-8% and in Figure 11 it is about 5%. As the location of the bottleneck moves downstream, the relative superiority of TKCS over CONWIP reduces. This is because cards in the upstream cells are more likely to be holding parts and waiting in their respective output hoppers.*

Figure 12: *Figure 12 is the counterpart of Figure 5 with machines that have Erlang processing time distributions instead of exponential distributions. Again, only the TKCS and the CONWIP configurations are plotted; TKCS is better by about 10-20%. The observations are similar to those made above.*

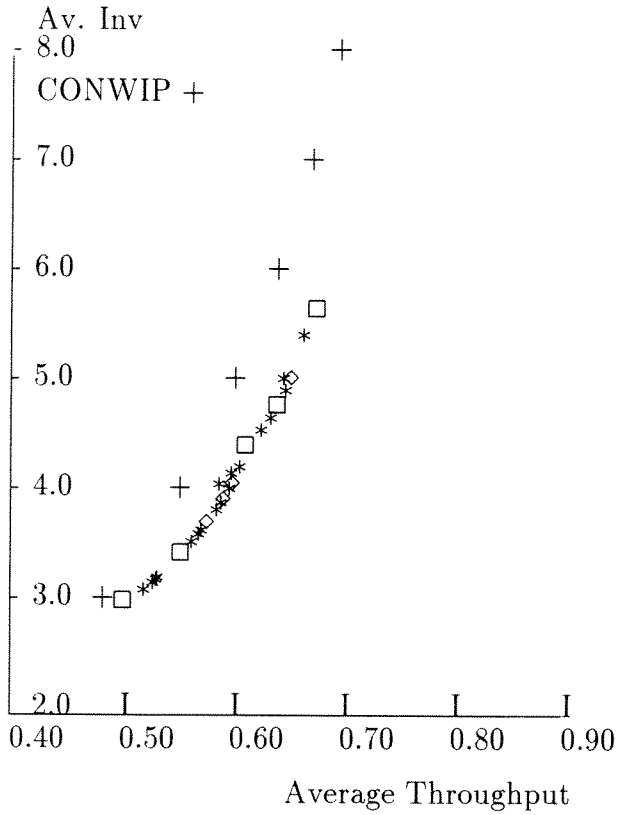


Figure 5: Average inventory vs. mean throughput for a 4 machine line with exponential machines and a bottleneck (exponential with mean 0.12) in the *first* position. The other three machines have a mean of 0.1.

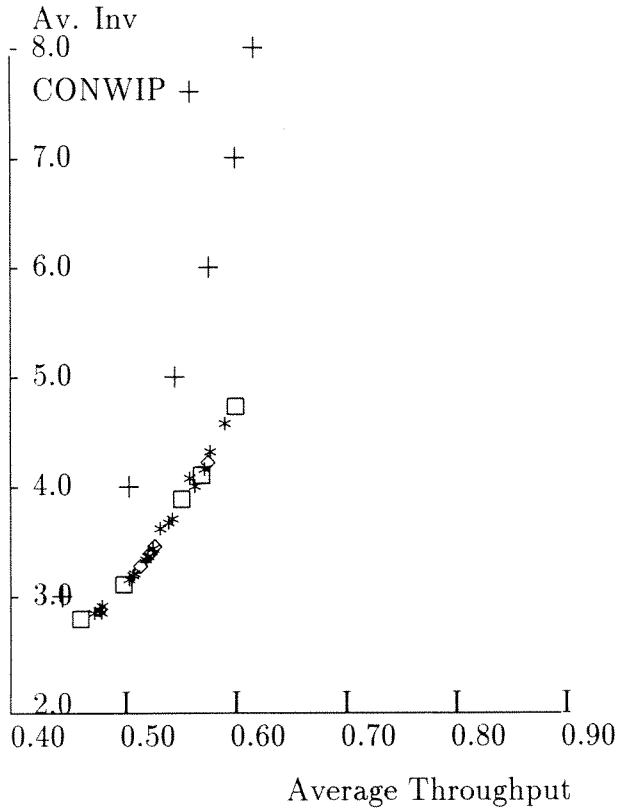


Figure 6: Average inventory vs. mean throughput for a 4 machine line with exponential machines and a bottleneck (exponential with mean 0.15) in the *first* position. The other three machines have a mean of 0.1.

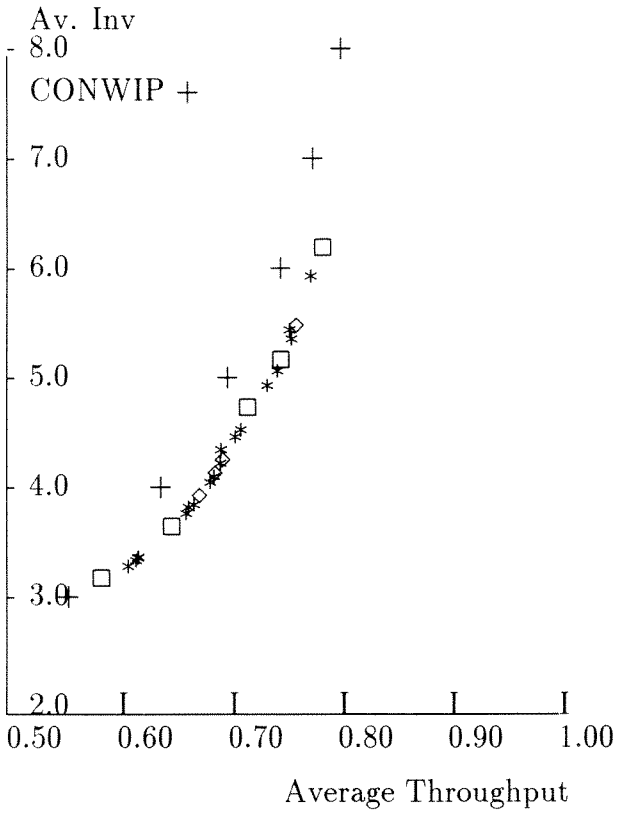


Figure 7: Average inventory vs. mean throughput for a 4 machine line with three Erlang(2) machines and with the *first* machine exponential. All machines have a mean processing time of 0.1.

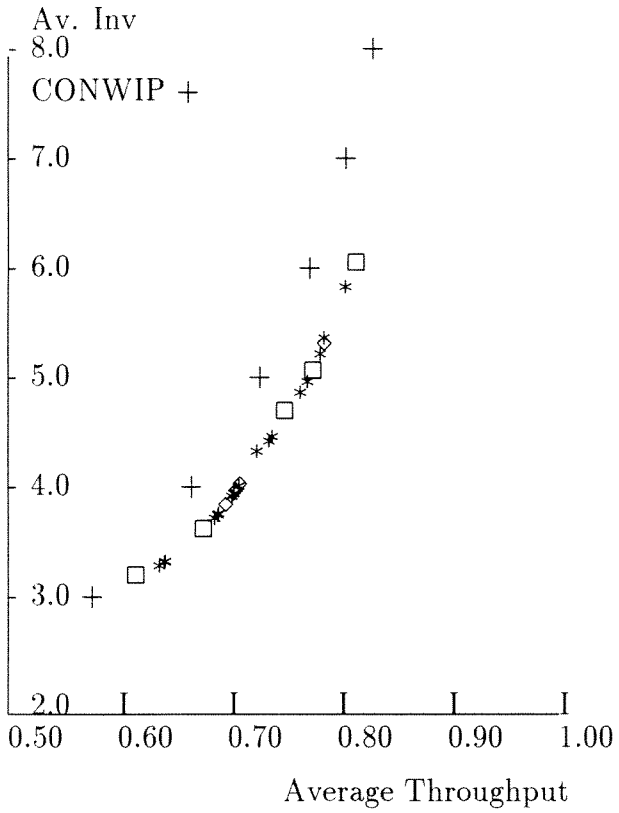


Figure 8: Average inventory vs. mean throughput for a 4 machine line with three Erlang(3) machines and with the *first* machine exponential. All machines have a mean processing time of 0.1.

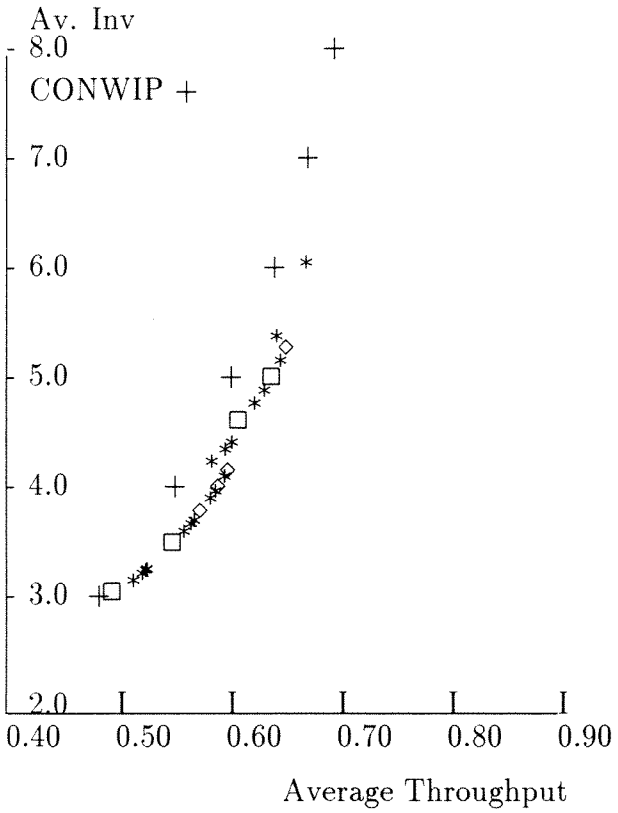


Figure 9: Average inventory vs. mean throughput for a 4 machine line with exponential machines and a bottleneck (exponential with mean 0.12) in the *second* position. The other machines have a mean of 0.1.

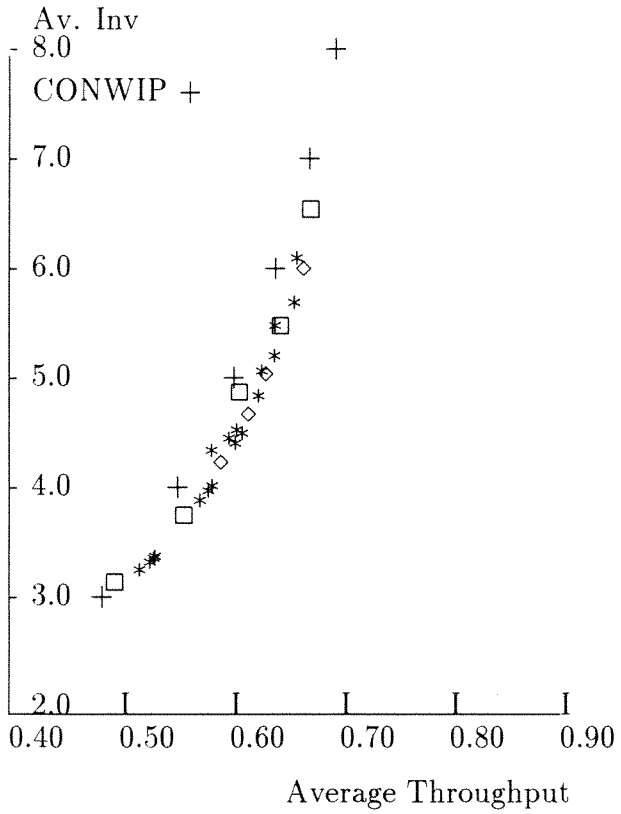


Figure 10: Average inventory vs. mean throughput for a 4 machine line with exponential machines and a bottleneck (exponential with mean 0.12) in the *third* position. All the other machines have a mean processing time of 0.1.

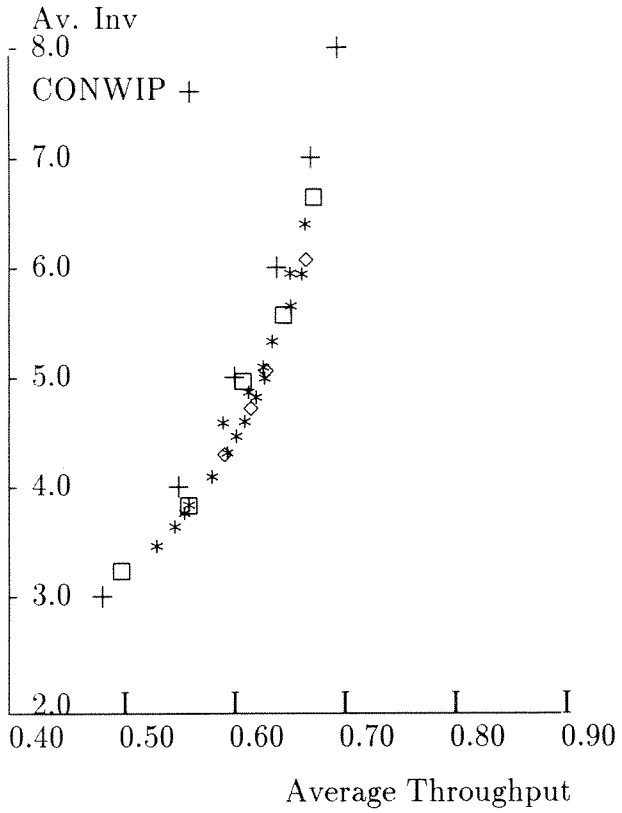


Figure 11: Average inventory vs. mean throughput for a 4 machine line with exponential machines and a bottleneck (exponential with mean 0.12) in the *fourth* position. The other machines have a mean processing time of 0.1.

The common aspect, however, is that TKCS is better than CONWIP with respect to *average* inventory. What is interesting is the observation that as the position of the bottleneck moves downstream, this superiority of TKCS diminishes. The more *severe* the bottleneck in a downstream location, the less superior is TKCS to CONWIP. Putting the above two observations together we conclude that if the bottleneck is severe and in the last position in the sequence, then TKCS is only marginally better than CONWIP (in *average* inventory). Finally, in all cases, as the desired average throughput increases, TKCS seems to perform better.

We also simulated lines of the following type: if the bottleneck is in location m (for example, $m = 1$ implies that the most upstream machine is the bottleneck) in a line with M machines, we use a $M - m + 1/(m, 1, \ldots, 1)/()$ partition. For the objective of *average inventory*, this rule provides a frontier that is not significantly different from a TKCS. We also simulated $M - m + 2/(m - 1, 1, \ldots, 1)$ and $3/(m - 1, 1, M - m)/()$ partitions for the cases when $m \geq 2$. These were only marginally inferior to the $M - m + 1/(m, 1, \ldots, 1)/()$ partition. Thus, if we have bottlenecks that shift upto position m (between position 1 and m), a robust efficient frontier can be obtained by selecting a $M - m + 1/(m, 1, \ldots, 1)/()$ partition. When variance data about the system are not known, a conservative approach is to have a CONWIP partition until bottlenecks are identified – as cards will collect at this machine. A starting design partition that we recommend, therefore, is a single cell provided it is feasible organizationally.

Before we conclude this subsection, we examine the objective of meeting the target mean throughput with minimum *variance* of the output. More precisely, the output of the line in any time interval (for example, a day or a week) is variable because of the randomness in the machine processing times, machine breakdowns and yield losses; we want to minimize this variation while meeting the mean target throughput (a long run measure). This becomes important in terms of meeting due dates effectively, and in being able to couple with other systems downstream. Computational testing shows that the output of a $N/(1, \ldots, 1)/(C_1, \ldots, C_N)$ partition has a *lower* coefficient of variation than the corresponding throughput in a CONWIP type configuration (see comment 2 in Appendix: Remarks).

To summarize, the partitioning of the line is crucial in order to achieve the efficient frontier when the trade-off is between average inventory and capacity. Furthermore, partitioning is also crucial in minimizing the coefficient of variation of the output process. A heuristic method to find a good partition for the objective of minimizing the average inventory is as follows: Find the most severe bottleneck in the sequence. If it is less than 10% slower than the rest, use TKCS. Otherwise, if its location is m, partition the line so that the first m machines are grouped in a cell, and the other machines are each in their separate cells.

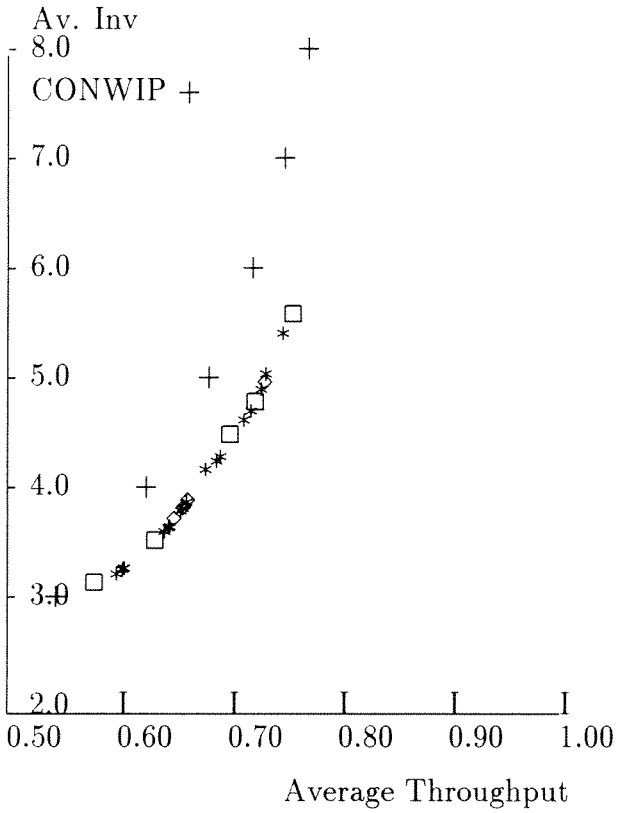


Figure 12: Average inventory vs. mean throughput for a 4 machine line with Erlang machines and a bottleneck (mean 0.12) in the *first* position. All other machines have a mean processing time of 0.1.

3.3 Sequencing

The third of our decisions is on the sequence of the machines. This may not be an option in many cases because of engineering reasons, or may be possible only among a subset of machines. It is possible at times to allocate extra effort (or to relocate people) to speed up processing or make engineering changes to reduce the yield losses. The sequencing results can be interpreted to determine where the maximum benefit can be obtained, and in what direction effort should be expended. As we have discussed yield losses in Muckstadt and Tayur (1993a), we will consider lines with no yield losses in our discussion here.

If the objective is to achieve a certain throughput with a given number of cards (the maximum inventory is fixed), a line and its reverse produce the same throughput. However, they may have different *average* inventories. In a TKCS, it is better to have the bottleneck in the first position than in the last. This is because if the bottleneck machine is last in the sequence, all the upstream cells have parts blocked in their output hoppers, whereas if the bottleneck machine is first in the sequence, then all the downstream cells have their cards posted on the bulletin board. Similarly, it is better to have the bottleneck in the second position than in the second from last position, and so on. Another interesting observation is that the bottleneck should be away from the center of the line in a TKCS because the interference to the working of a machine (by blocking and starving) is highest in the center of the line. Hiller and Boling(1966), among others, have observed this in simulation experiments. The mathematical proof for this phenomenon has been provided for certain special and simple cases (Rao(1976), Shanthikumar et al.(1991)), but the general case is still an open problem. Note that in CONWIP the sequence of the machines is unimportant as long as there are no yield losses. This can be seen from comparing the top curves in Figures 6-12. In the case where the sequence is fixed due to engineering reasons, one can move workers from the upstream location to the corresponding downstream bottleneck location, or speed things up in the middle of the line first rather than at the ends.

For partitions other than CONWIP and TKCS, we need the following descriptive phrases. The *load* of a partition is $\sum_{i=1}^{N} M_i^2$. If M_i is larger upstream than downstream, we have *front* loading; analogously we have *rear* loading. Suppose we have two potential partitions of a line: 2/(a,b)/() and 2/(b,a)/() with a>b. As the target throughput increases, the latter partition has a lower average inventory. To generalize, given a partition and its reverse, at higher target throughput levels the one with a lesser *front* loading is preferred. Between *partitions*, the one with the smallest *loading* is preferred with respect to average inventory. Thus, as in the previous section we prefer TKCS, as TKCS has a smaller *load* and CONWIP the *highest.*

Recall that when dealing with a line that has yield losses at some processing stages, the sequence of operations is critical. A general rule of thumb regarding the sequence is: Reduce yield losses in processes from downstream first to upstream, and concentrate at the center of the line first to increase the mean and reduce the variance of processing times.

4 Summary

Having described a number of general results, we briefly summarize the results of Muckstadt and Tayur(1993) and this paper.

1. If I don't have flexibility in choosing the number of machines in a cell (due to space constraints, organizational structure, or layout), then how do I distribute the cards to the cells? How many cards do I need?

   Pick $C \in \{N, N+1, \ldots\}$, and for each value use the pulse and state-space heuristics to determine the best allocation. Simulate. Pick the C that achieves the target.

2. If I have complete freedom to choose the number of cells and the number of machines I can place in them, then should I put them all in one cell or should I put exactly one machine in every cell and thus have many cells?

   If your objective is to minimize the number of cards, then have only one cell. If the objective is to minimize average inventory, place exactly one machine in each cell.

3. If I have a restriction on the total number of cards, then how do I configure the line?

   If you can achieve a target throughput by a M/(1, ... ,1)/(), do it. Else, have a $k/(1, \ldots 1, M-k+1)/()$ line, with k as large as possible.

4. If I can partially alter the sequence in which the operations (machines) are performed, then what is a good sequence to select?

   If there are yield losses, use the intuition developed in Muckstadt and Tayur (1993). If not, have the operations with high variance and/or high mean processing times away from the center of the line, preferably upstream.

5. If I have a bottleneck, how do I manage it? Is the location of the bottleneck in the sequence of machines important?

   The location is important. If it is in location m, then use a M-m+1/(m, 1 ... ,1)/() strategy.

6. I have some machines that breakdown, while others have wide variance in processing times. How is the control strategy affected by these two different sources of variability?

   It isn't. These are different manifestations of variance that have the similar effect on structural properties (and differ only in degree).

7. Some processes require rework, while others cause scrap. How do I design the system to account for these?

   Rework is equivalent to processing time variation. Use the same strategy as above. Yield loss is different; sequence the machines appropriately as discussed earlier. Observe that the kanban mechanism will react dynamically to these variabilities.

8. I have shifting bottlenecks. What is a robust control strategy?

   If the bottlenecks shift through operation m, then use M-m+1/(m, 1 ... ,1)/() strategy.

9. Currently, I do not have data on the variances of processing times but I am collecting them now. In the mean time, how do I run my line? Should I change once I know the values of the variances?

   Use 1/M/(C), also known as CONWIP, when you don't know the values of the variances; however, once you do, the choice of the best strategy depends on your objective (see 2-5 above).

10. I have no data on the parameters of the machines (speed, uptime, downtime). What control mechanism is best suited to identify my bottlenecks and problem areas, while keeping me afloat in the short term?

    CONWIP, again.

11. I have short term targets. I want to be a reliable supplier, and so I wish to reduce the variance of my output. What design ensures the most reliable service?

    $N/(1,\ldots,1)/()$.

12. I want to move toward zero inventory. Shouldn't I simply reduce the total number of cards in the system? Does the control strategy I choose affect my ability to achieve my goals?

    This will cause you to change the configuration so that you can meet the throughput, if the parameters on the machines are not improved. Once you change the configuration without improving the machine parameters, the average inventory will actually go up.

As considerable research has been done on buffering of serial production lines taking into account manufacturing uncertainities, the purpose of this paper has been to summarize, unify and extend some of the previous results, and provide some new insights. This stream of research is aimed at gaining insight into complex manufacturing scenarios that are intractable by classical analytical models and techniques. In particular:

- We have considered three objectives (while meeting the target mean throughput rate): minimize the maximum inventory, minimize the average inventory, and minimize the coefficient of variation of the output.
- We have considered the management of bottleneck machines.
- We have provided intuition and rules for the design of serial lines: sequencing of machines (or equivalently where to allocate or relocate effort), partitioning into cells and allocation of kanbans.

Future work deals with demand variation and correlation aspects in a multi-product setting. Much work still needs done to understand the interactions between the effects of demand variability and production uncertainities.

allocation	mean throughput	average inventory	allocation	mean throughput	average inventory
(1,5)	0.6619	5.208	(1,2,3,1)	0.6710	5.183
(1,7)	0.7260	6.756	(1,3,4,1)	0.7294	6.704

Table 3: Comparing 2/(1,3)/() with 4/(1,1,1,1)/().

Appendix: Remarks

1. The computation of average inventory can be done differently in some cases. Consider a 1/(2)/(C) line. Instead of allowing the raw-material to enter the system as soon as a card arrives on the bulletin board, we could wait further until the queue at the first machine is empty. The throughput remains unchanged, but the average inventory drops (and is equal to that of a 2/(1,1)/(1,C-1) line). Prima-facie, it may appear that our results regarding the differences between CONWIP and TKCS are simply an accounting artifact; however, in lines with more than three machines, there is a difference in the average inventories between 1/(M)/(C) and M/(1, ...,1)/() at equal mean throughputs, even under the new accounting procedure. Table 3 shows a typical result that is obtained by comparing a 2/(1,3)/() partition with a 4/(1,1,1,1)/() partition; in this example, all the machines have an exponential distribution with mean 1. Note that for $M \geq 3$, the mean throughput of a 2/(1,M)/(1,C-1) is lower than a 1/(M)/(C) line (in a general case, barring trivialities), and the inventory is not constant. In fact, a systematic study conducted indicated that as we move from CONWIP (1/(M)/()) to 2/(1,M-1)/() to 3/(1,1,M-2)/() and so on to M/(1,..., 1)/(), the average inventory monotonically reduces at equal mean throughputs. A conjecture for this phenomenon is that material release into the system is delayed, and opportunities for a lot of WIP to accumulate is limited.

2. The output of a serial production line in any interval of time is stochastic, and so researchers distinguish between the mean throughput (capacity) and throughput (a random variable). Thus, we can examine the variance of throughput, etc. A study of a four machine line was conducted. The ratio of the coefficient of variation (c.v.) of the output of CONWIP to the c.v. of the output of a TKCS was tabulated (at equal mean throughputs for a range of mean throughputs) for balanced lines with machines having exponential distributions, and for lines with a bottleneck. On an average the ratio was 1.47, with a high of 1.53 and a low of 1.38 in our experiments.

3. The study of average flow times is equivalent to study of average inventories (by virtue of Little's law) in certain queuing systems. Current research is focussed on showing Little's result in multi-stage kanban lines.

4. Combinatorial properties of our heuristics are studied in Tayur(1992). Coupled with properties of throughput, these provide additional computational savings. Briefly, we exploit the Schur-concavity of the throughput (random variable) with respect to kanbans to reduce the search for optimal allocations.

sequence	allocation	throughput
ABAA	(1,2,2,1)	0.6426
	(1,3,1,1)	0.6146
	(1,3,3,1)	0.6923
	(1,4,2,1)	0.6825
	(1,3,2,1)	0.6598
	(1,2,3,1)	0.6570
BAAA	(1,2,2,1)	0.6417
	(1,3,1,1)	0.6139
ACAA	(1,2,2,1)	0.6258
	(1,3,1,1)	0.6036
CAAA	(1,2,2,1)	0.6267
	(1,3,1,1)	0.6044

Table 4: A little imbalance in the line does not change the optimal allocation.

5. A discrete time version of this kanban model can be compared to a system operating with order up-to policies. Note that there are very limited theoretical results in multi-stage, stochastic, capacitated systems operated by base-stock policies. An Infintisimal Perturbation Analysis (IPA) based approach is considered in Glasserman and Tayur(1992) for a multi-stage system in discrete time.

6. As most lines in the real world are rarely perfectly balanced, a simulation experiment was conducted by changing the mean of some of the machines by about 5%. The optimal allocations coincided with those obtained for a perfectly balanced line. Table 3 provides results for some different imbalances in a 4/(1,1,1,1)/() line. Machine A has a exponential distribution with rate 1, machine B has an exponential distribution with rate 1.05, and machine C has an exponential distribution with rate 0.95. The results are exact and have been obtained by solving the balance equations.

7. The simulations for Figures 3-12 and Tables 1-4 were conducted in the following manner. For experiments regarding mean throughput and average inventory, data were collected after every 1000 outputs from the lines, and the intialization bias was accounted for by rejecting the first 1000 outputs (a pilot study was done to determine that 1000 was sufficient). The simulation was run for a total of 20000 outputs. A batched mean estimator was used. The confidence interval obtained due to this procedure was within $\pm$ 0.5% of the nominal value; as an example, for the first entry in column 2 in Table 2 (0.4504), the confidence interval obtained was $\pm$ 0.0015. For collecting results for the output of the line (remark 2), after the initialzation bias was accounted for, the number of outputs in a fixed interval was tabulated. Again, this was done for 20 consecutive periods, and a batched mean estimator was used. The simulation was set up to include synchronization using different streams of random variables and antithetic variates for variance reduction. Note that only a small set of simulations conducted are shown in the figures and tables in this paper. As an example, in Figure 3 each of the 44 points shown is a simulation instance, and only represent points generated from

two of the many partitions simulated.

8. A program that solves the balance equations exactly was also developed. This can solve the case of exponential processing times. This was used to validate the simulation.

References

[1] Conway, R., W. Maxwell, J. McClain, and L. J. Thomas. *The Role of Work-in-process Inventory in Serial Production Systems*, Oper. Res., vol. 36, no. 2, pp. 229-241, 1988.

[2] Deleersnyder, Jean-Luc, Thom J. Hodgson, Henri Muller, and Peter O' Grady. *Pull Systems: An Analytic Approach*, Mgmt. Sci., vol. 35, no. 9, , pp. 1079-1091, 1989.

[3] Glasserman, P. and S. Tayur. *Sensitivity Analysis for Base Stock Levels in Multi-Echelon Production Inventory Systems*, WP 1992-27, GSIA, CMU, 1992.

[4] Hillier, F. S. and R. W. Boling. *The Effect of Some Design Factors on the Efficiency of Production Lines with Variable Operations Times*, The Journal of Industrial Engineering, vol. 17, no. 12, 1966.

[5] Mitra, D., and I. Mitrani. *Analysis of a Kanban Discipline for Cell Coordination in Production Lines, 1.*, Mgmt. Sci., vol.36, no. 12, pp. 1548-1566, 1990.

[6] Muckstadt, J. and S. Tayur. *A Comparison of Alternative Kanban Control Mechanisms: Background and Structural Results*, GSIA Working Paper, May 1993.

[7] Rao, N.P. *A Generalization of the Bowl Phenomenon in Series Production Systems*, Int. J. Oper. Res., vol. 14, no. 4, pp. 437-443, 1976.

[8] Shanthikumar, J.G., G. Yamazaki and H. Sakasegawa. *Characterization of Optimal Order of Servers in a Tandem Queue with Blocking*, OR Letters (10), pp. 17-22, 1991.

[9] Tayur, S. *Analysis of a Kanban Controlled Serial Manufacturing System.* Ph.D Thesis, Cornell University, 1990.

[10] Tayur, S. *Structural Results and a Heuristic for Kanban Controlled Serial Lines*, Technical Report No. 934, SORIE, Cornell University, (revised) May 1992.

[11] Woodruff, D. L., M. L. Spearman and W. J. Hopp. *CONWIP: A Pull Alternative to Kanban*, Int. J. Prod. Res., vol. 28, no. 5, pp.879-894, 1990.