Factors Affecting Opportunity of Worksharing as a Dynamic Line Balancing Mechanism

Esma S. Gel
Department of Industrial Engineering,
Arizona State University, Tempe, AZ 85287-5906

Wallace J. Hopp
Department of Industrial Engineering and Management Sciences
Northwestern University, Evanston, IL 60208-3119

and

Mark P. Van Oyen
Department of Information Systems and Operations Management,
Loyola University Chicago, Chicago, IL 60611-2196

June 12, 2000

Corresponding Author: Esma S. Gel
Dept. of Industrial Engineering, Arizona State University,
P.O. Box 875906, Tempe, AZ 85287-5906
Ph: 480-965-2906, FAX: 480-965-8692
E-mail: esma.gel@asu.edu

Factors Affecting Opportunity of Worksharing as a Dynamic Line Balancing Mechanism

Esma S. Gel
Department of Industrial Engineering,
Arizona State University, Tempe, AZ 85287-5906

Wallace J. Hopp
Department of Industrial Engineering and Management Sciences
Northwestern University, Evanston, IL 60208-3119

and

Mark P. Van Oyen
Department of Information Systems and Operations Management,
Loyola University Chicago, Chicago, IL 60611-2196

Abstract

We consider the problem of optimal worksharing between two adjacent workers each of whom processes a fixed task in addition to their shared task(s). We use a Markov Decision Process (MDP) model to compute optimal policies and provide a benchmark for evaluating threshold policy heuristics. Our approach differs from previous studies of dynamic line balancing in that we focus on system architecture factors that affect the performance improvement opportunity possible through worksharing relative to a traditional static worker allocations, as well as practical heuristics for worksharing. We find that three such factors are significant whether we use an optimal or a heuristic control policy: ability to preempt the shared task, granularity of the shared task and overall variability of the task times. Understanding the role of these factors in a given production environment provides a means for determining where and how worksharing can have significant logistical benefits.

1 Introduction

The basic objective of line balancing is to achieve efficient utilization of productive capacity. The focus of classical industrial engineering studies of line balancing has been on evenly distributing work in the line over the long term. But even if a line is balanced with regard to average loads it can still become seriously unbalanced in the short term due to variability. Changes in product mix, machine failures, quality problems, operator related issues and many other events can cause workloads at individual stations to become temporarily imbalanced. Traditionally, this variability has been buffered by either capacity (e.g., the line is paced to allow operators 60 seconds to complete their tasks, even though the tasks require only 45 seconds on average to complete) or inventory (e.g., WIP in buffers between stations, which serves to partially decouple the stations from one another). In recent years, however, spurred by the JIT and lean manufacturing movements, another alternative for buffering variability in a flowline has arisen, *workforce agility*. By having cross-trained workers perform multiple tasks, an agile worksystem can shift work or workers from one station to another

to balance the load in the short term and thereby increase the effective capacity. In the most extreme case, which we call full cross-training, every worker is trained at every operation in the line and ample equipment is available to prevent workers from blocking each other. Such systems achieve line balancing and can dramatically increase system performance, as we showed in [10]. Studies of full cross-training systems can be found in several other papers, including [1], [3], [11]. Our focus in this paper, however, is to investigate systems that require only partial cross-training of the workforce.

[6] examined a specific type of workforce agility with partial cross-training that they refer to as *dynamic line balancing*. For a flow line with interstation buffers, they assumed: (1) each worker is assigned to one *fixed task* as well as one or two *shared tasks* that could also be done by either their upstream or downstream neighbor, (2) once an item is transferred to a downstream workstation, it cannot return, (3) a workstation may withdraw items from its upstream buffer in any order (i.e., picking the job with the shortest mean processing time is allowed), (4) a worker becomes blocked if there is no space in his/her downstream buffer to transfer his/her finished item, (5) a worker becomes starved if his/her upstream buffer is empty. The authors emphasized that buffers serve two main purposes: providing work for the downstream stage and providing storage space for the upstream stage. Based on this observation, they proposed the so-called SPT/RSPT family of policies under which the following two rules are applied: (1) SPT: a worker selects the shortest job from his upstream buffer, (2) RSPT: a worker processes the shared task on a job if and only if his/her downstream buffer has less than R "long" jobs (i.e., jobs for which the shared task has not been performed). To evaluate different dynamic line balancing strategies, the authors used the performance metric of efficiency, which they defined as actual throughput rate divided by maximum possible throughput (calculated by setting all worker utilizations equal to 100%).

The majority of the analysis in [6] considered two station lines with exponential processing times. Using Markov chain models, the authors demonstrated the effectiveness of the SPT/RSPT type policies and showed that setting the cutoff level R to half of the buffer size works well, particularly when the fraction of the shared task to total work is around 40%. In addition to two station lines, they made simulation studies of longer lines (5 stations) and smaller processing time variability, which echoed their findings for two station lines with exponential processing times.

In a technical note subsequent to [6], [5] modeled each task as a set of subtasks and proposed new rules that measure the number of subtasks not yet completed on jobs in the downstream station (in the buffer and in process), instead of just counting the number of long tasks. The authors assumed that the subtasks are identically distributed exponential random variables with unit mean. Hence, counting the number of subtasks gives a precise measure of how much work

is available for the downstream worker. Their analysis included simulation studies of two-stage lines under various scenarios, and some of three- and five-stage lines. The authors concluded that dynamic line balancing can be used effectively for low-inventory systems in which the material handling system requires jobs to be processed first-come-first-served (FCFS). Their analysis also confirmed that the simple rule of sending a shared task downstream if the buffer is less than half-full, is effective.

The above cited papers on dynamic line balancing, as well as most of the other studies on worksharing, have assumed a fixed architecture for the system and have focused on establishing the effectiveness of a particular family of policies. While this is an important and logical place to start studying workforce agility, in the long term system architecture is not fixed. For example, changing the material handling system from a conveyor with unidirectional flow of units, which prevents any shared tasks from being sent upstream, to a carousel type system which allows all workers to drop and pick up tasks in any order, significantly alters the possibilities for worksharing.

In this paper, we study dynamic line balancing to address some key issues by taking an approach different than the ones taken in previous work on the subject. First, we take a control-oriented approach to the problem of worksharing between two workers rather than assuming a certain structure (such as a threshold property) for the worksharing policy. Specifically, we develop Markov Decision Process (MDP) formulations for simple two-station models, the solutions of which lead us to new insights into the complex structure of an optimal policy, as well as an evaluation of practical threshold policy heuristics. To our knowledge, our results are the first of this type. Secondly, we investigate how system architecture affects the logistical benefits possible through worksharing. We are particularly interested in how the following two metrics depend on the system architecture: (1) *opportunity*, which measures the maximum performance improvement through worksharing (i.e., how well the optimal worksharing policy performs relative to the best static policy), and (2) *percent suboptimality*, which measures the effectiveness of a heuristic in a particular system architecture (i.e., how well a heuristic performs relative to the optimal worksharing policy).

The key contributions of this paper are to provide insights into effective control of worksharing at the operational level, as well as recommendations on the design of a physical environment that will enhance the performance improvement workforce agility offers. For this purpose, we explore the effect of three system architecture factors: (1) preemption of the shared task (i.e., whether a task that is already started can be transferred to the downstream station at any point of processing), (2) granularity of the shared task (i.e., whether a nonpreemptive shared task can be transferred at certain break points during processing), and (3) processing time variability.

The remainder of the paper is organized as follows. Section 2 includes our modeling framework

in terms of a detailed description of the system and the basic MDP formulation of the optimal worksharing problem. In Section 4, the optimal performance (throughput) computed from this model is used to demonstrate the trade-off between WIP and worker flexibility. Section 3 presents definitions of the heuristics we have observed to be effective as well as their comparisons to the optimal policy under preemptive and nonpreemptive shared task assumptions. Finally, in Section 5 we study the effects of job preemption, work content granularity and task processing time variability.

2 Modeling Framework

We consider two stations in tandem attended by two workers, worker 1 (W1) at station 1 and worker 2 (W2) at station 2. The system runs under the CONWIP discipline with a WIP level of M jobs. The buffers in front of the two stations (buffer 1 and buffer 2) have no capacity limitations except for the CONWIP level of the system.

Each job has 3 consecutive tasks to be performed, tasks A, B and C. A job is called *type A* (*type B*, *type C*) if the task to be completed next on the job is task A (B, C). Task A can only be performed by W1 at station 1 and task C can only be performed by W2 at station 2, whereas task B can be performed by either worker at either station. We assume that there is sufficient tooling/machinery at the stations (or at a third station which the workers can access in a negligible amount of time) for simultaneous task B processing on different jobs. Jobs are processed in first come first served (FCFS) order at both stations. Processing times of task A, task B and task C of job l (denoted by $S_{\mathsf{A}}(l)$, $S_{\mathsf{B}}(l)$ and $S_{\mathsf{C}}(l)$, $l = 1, 2, \dots$) are mutually independent, exponentially distributed with rates μ_{A}, μ_{B} and μ_{C}, respectively. We define T_{A}, T_{B}, T_{C} as the mean processing times of the tasks, A, B and C, respectively. We assume that task B processing by W1 can be preempted but processing by W2 cannot be interrupted, and hence once a B task is in buffer 2, it has to be processed to completion by W2. This preemption structure is consistent with a unidirectional flow of materials facilitated by a material handling system. We call this the "preemptive shared task" assumption, and will later disallow it to study the effects of preemption.

We allow workers to have different processing speeds and for that purpose define v_2 as the speed of W2 relative to that of W1. That is, along a sample path $\omega \in \Omega$, the processing of the B task of job l will take $S_{\mathsf{B}}(l, \omega)$ time units if processed by W1 and $S_{\mathsf{B}}(l, \omega)/v_2$ time units if processed by W2. This way of modeling workers presents a limitation in that worker speeds may in practice vary as a function of the system state instead of being stationary over time, particularly in labor intensive low-inventory systems in which workers can be observed to be the cause of idle time. [8] studied this phenomenon and concluded that processing time distributions of labor intensive tasks (such as entering data into a computer as in their experiment) are not independent of the size of the buffer, the processing speeds of co-workers, or the amount of inventory in the system. In a subsequent

study, [7] developed mathematical models to study workers that change their processing times as a function of system state and how it effects the efficiency of serial lines. They concluded that workers who speed up to prevent blocking and starving can substantially improve performance. While our assumption may be an incomplete representation of reality for some systems, we believe that our results are still generally valid for many systems in which worker skill is the dominant determinant of the speed of processing or the intentional use of state-dependent worker speeds is disallowed due to union rules and/or organizational culture.

The state of the system at time t is described by the vector

$$Z(t) = (X(t), Y_{\mathsf{M}}(t), Y_{\mathsf{M}-1}(t), \ldots, Y_1(t)) \ ,$$

where $X(t)$ denotes the task type in process at W1, and $Y_{\mathsf{i}}(t)$, for all $i = 1, 2, \ldots, M$, denotes the task type of the i^{th} job in buffer 2 at time t. Specifically, $Y_1(t)$ denotes the task type in process at station 2 at time t. The following summarizes how the system state is defined.

$$X(t) = \begin{cases} A & \text{if W1 is processing a type A job,} \\ B & \text{if W1 is processing a type B job,} \\ O & \text{if all } M \text{ jobs are at buffer 2,} \end{cases} \tag{2.1}$$

$$Y_{\mathsf{i}}(t) = \begin{cases} B & \text{if } i^{\text{th}} \text{ job is type B,} \\ C & \text{if } i^{\text{th}} \text{ job is type C,} \\ O & \text{if location } i \text{ in buffer 2 is empty.} \end{cases} \tag{2.2}$$

For example, state $(BOOCBC)$ in a system with 5 jobs ($M = 5$) denotes that W1 is processing a B task, W2 is processing a C task, one type B job and one type C job are in buffer 2, and (implicitly) there is one type A job awaiting service in buffer 1. Note that the state of the system denotes not only the quantity but also the sequence and type of jobs in buffer 2. Although the size of the state space, $\mathcal{S}$, looks like $3^{\mathsf{M}+1}$ at first glance, it is in fact $2^{\mathsf{M}} + \sum_{\mathsf{i}=1}^{\mathsf{M}} 2^{\mathsf{i}}$ due to the infeasibility of some states.

Decision epochs occur whenever W1 completes an A task. A policy g specifies that W1 either continues working on the new type B job (*keep*), or transfers it to buffer 2 to await service by W2 (*put*). Let $\mathcal{U} = \{keep, put\}$ denote the action space. The set of states for which an action is defined includes all states in which W1 is processing a type B job: $\mathcal{S}_{\mathcal{U}} = \{(x, y_{\mathsf{M}}, y_{\mathsf{M}-1}, \ldots, y_1) \in \mathcal{S} : x = B\}$. Equivalently, one can restrict the action to *keep* at all other states. The long run average throughput rate of policy $g(z) \in \mathcal{U}$ for all $z \in \mathcal{S}$, is expressed as

$$\bar{J}_{\mathsf{g}} = \lim_{\mathsf{T}\to\infty} \frac{1}{T} \, E\left\{ \int_0^{\mathsf{T}} \mathbb{1}\{Y_1(t) = C\} \, \mu_{\mathsf{C}} \, dt \right\} \ . \tag{2.3}$$

The class of admissible policies, G, is taken to be the set of stationary, non-anticipative policies and the set of decision epochs is assumed to be the set of all task completion times, $t_{\mathsf{k}}, \ k = 1, 2, \ldots,$

at which $X(t_k) = B$. The objective of the optimization problem is to determine a policy $g^* \in G$ that maximizes $\bar{J}(g)$.

The optimal control problem considered here can be formulated as a discrete-time stochastic dynamic programming problem by using uniformization [4]. The uniformized version has a transition rate $\gamma = \text{maximum}\{\mu_A + v_2\mu_B, \mu_A + \mu_C, \mu_B + v_2\mu_B, \mu_B + \mu_C\}$ for all states. Let $V_k(z)$ denote the maximum expected output rate over the next k stages. With the initial condition that $V_0(z) = 0$ for all $z \in \mathcal{S}$, the DP equation of the discrete time model is given as follows (for all $z = (x, y_M, y_{M-1}, \ldots, y_1) \in \mathcal{S}$):

$$\begin{aligned} V_k(A, y_M, y_{M-1}, \ldots, y_1) &= \mathbb{1}\{y_1 = C\}\mu_C + \frac{\mu_A}{\gamma} V_{k-1}(C_1(z)) \\ &\quad + \mathbb{1}\{y_1 = B\}\frac{v_2\mu_B}{\gamma} V_{k-1}(C_2(z)) \\ &\quad + \mathbb{1}\{y_1 = C\}\frac{\mu_C}{\gamma} V_{k-1}(D_2(z)) \\ &\quad + \left(1 - \frac{\mu_A}{\gamma} - \mathbb{1}\{y_1 = B\}\frac{v_2\mu_B}{\gamma} \right. \\ &\qquad \left. - \mathbb{1}\{y_1 = C\}\frac{\mu_C}{\gamma}\right) V_{k-1}(z) \ , \end{aligned} \tag{2.4}$$

$$\begin{aligned} V_k(O, y_M, y_{M-1}, \ldots, y_1) &= \mathbb{1}\{y_1 = C\}\mu_C + \mathbb{1}\{y_1 = B\}\frac{v_2\mu_B}{\gamma} V_{k-1}(C_2(j)) \\ &\quad + \mathbb{1}\{y_1 = C\}\frac{\mu_C}{\gamma} V_{k-1}(D_2(j)) \\ &\quad + \left(1 - \mathbb{1}\{y_1 = B\}\frac{v_2\mu_B}{\gamma} \right. \\ &\qquad \left. - \mathbb{1}\{y_1 = C\}\frac{\mu_C}{\gamma}\right) V_{k-1}(j) \ , \end{aligned} \tag{2.5}$$

$$\begin{aligned} V_k(B, y_M, y_{M-1}, \ldots, y_1) &= \max\{K_k(B, y_M, y_{M-1}, \ldots, y_1) \ , \\ &\qquad P_k(B, y_M, y_{M-1}, \ldots, y_1)\} \ , \\ &= \max\{K_k(z), V_k(T_1(z))\} \ , \end{aligned} \tag{2.6}$$

where the function $K_k(z)$ is defined as

$$\begin{aligned} K_k(B, y_M, y_{M-1}, \ldots, y_1) &= \mathbb{1}\{y_1 = C\}\mu_C + \frac{\mu_B}{\gamma} V_{k-1}(D_1(z)) \\ &\quad + \mathbb{1}\{y_1 = B\}\frac{v_2\mu_B}{\gamma} V_{k-1}(C_2(z)) \\ &\quad + \mathbb{1}\{y_1 = C\}\frac{\mu_C}{\gamma} V_{k-1}(D_2(z)) \\ &\quad + \left(1 - \frac{\mu_B}{\gamma} - \mathbb{1}\{y_1 = B\}\frac{v_2\mu_B}{\gamma} \right. \\ &\qquad \left. - \mathbb{1}\{y_1 = C\}\frac{\mu_C}{\gamma}\right) V_{k-1}(z) \ . \end{aligned} \tag{2.7}$$

In this formulation, operator T_1 describes the transfer of a type B job into buffer 2. Operators C_1 and C_2 denote task A completion at station 1 and task B completion at station 2, respectively. Operators D_1 and D_2 describe the departure of a type B job from station 1 and the departure of a type C job from station 2 upon completion, respectively. Formally, the operators T_1, C_1, C_2, D_1 and D_2 are defined as follows for $i \in \{M-1, M-2, \ldots, 1\}$.

$$T_1(z) = \begin{cases} (A, O, \ldots, O, B, y_{i-1}, y_{i-2}, \ldots, y_1) \\ \qquad \text{for all } (z : x = B;\ y_M, y_{M-1}, \ldots, y_i = O; \\ \qquad\qquad y_{i-1}, y_{i-2}, \ldots, y_1 \neq O), \\ (O, B, y_{M-1}, y_{M-2}, \ldots, y_M) \\ \qquad \text{for all } (z : x = B;\ y_M = O; y_{M-1}, y_{M-2}, \ldots, y_1 \neq O). \end{cases} \tag{2.8}$$

$$C_1(z) = (B, y_M, y_{M-1}, \ldots, y_1), \qquad \text{for all } (z : x = A). \tag{2.9}$$

$$C_2(z) = (x, y_M, y_{M-1}, \ldots, C), \qquad \text{for all } (z : y_1 = B). \tag{2.10}$$

$$D_1(z) = \begin{cases} (A, O, \ldots, O, C, y_{i-1}, y_{i-2}, \ldots, y_1) \\ \qquad \text{for all } (z : x = B;\ y_M, y_{M-1}, \ldots, y_i = O; \\ \qquad\qquad y_{i-1}, y_{i-2}, \ldots, y_1 \neq O), \\ (O, C, y_{M-1}, y_{M-2}, \ldots, y_M) \\ \qquad \text{for all } (z : x = B;\ y_M = O; y_{M-1}, y_{M-2}, \ldots, y_1 \neq O). \end{cases} \tag{2.11}$$

$$D_2(z) = \begin{cases} (x, O, \ldots, O, O, y_{i-1}, y_{i-2}, \ldots, y_2) \\ \qquad \text{for all } (z : y_1 = C;\ y_M, y_{M-1}, \ldots, y_i = O; \\ \qquad\qquad y_{i-1}, y_{i-2}, \ldots, y_1 \neq O), \\ (A, O, y_M, y_{M-1}, \ldots, y_2) \\ \qquad \text{for all } (z : y_1 = C; y_M, y_{M-1}, y_{M-2}, \ldots, y_1 \neq O). \end{cases} \tag{2.12}$$

Unfortunately, the optimal policy for the MDP model does not have a simple structure. The complexity of the optimal policy makes it impractical to use in real life systems, especially since dynamic line balancing rules must be applied by workers in real time. For example, consider our problem with a WIP level of 10 jobs, which means that there are 1023 states for which the workers need to know the proper action. Without a simple structure, we would have to communicate this with a list that specifies either *keep* or *put* for each of these states. To be practical, we probably need to restrict attention to *threshold* policies in which the action is chosen by comparing the current level of a one-dimensional system state measure to a threshold level.

In our numerical tests we have encountered many problem instances that strongly suggest that a threshold policy is not optimal in general. This is due to the second order effects of variability and the sequence of the tasks in the buffer. For example, an intuitive one-dimensional measure for sorting states and determining a threshold level is the mean work content in buffer 2. But the states $(BOOCBB)$, $(BOOBCB)$ and $(BOOBBC)$ would be indistinguishable according to this sorting

method since all of them have two type B and one type C jobs in buffer 2. If $T_{\mathsf{C}} = 2\ T_{\mathsf{B}}$, then $(BOOOCC)$ is also an equivalent state. However, one can easily think of reasons why the optimal actions in these states could be different. For instance, compare the likelihood of an imminent starvation for W1 at states $(BOOBCB)$ and $(BOOBBC)$, given that the action of transferring the B task to the downstream station is selected. Clearly, this likelihood is higher in state $(BOOBCB)$ and hence, an optimal policy must be able to differentiate between these states for certain values of the system parameters, μ_{A}, μ_{B} and μ_{C}.

To account for these two factors and find a threshold policy that is optimal among the set of all feasible policies, we have tried several sorting algorithms including ones that consider the probability that W1 will finish the tasks in buffer 1 before W2 finishes the tasks in buffer 2. Unfortunately, for each we found an example for which the optimal policy violated the threshold property. Hence, a threshold policy such as the one described in [6] and [5] is not necessarily optimal. However, since we know that practicality constraints limit us to threshold policies, the important questions to address are: (1) what is the best threshold policy? and (2) how does it perform?

The MDP model provides a numerical baseline for evaluating the quality of heuristics based on the threshold property. In addition, it allows us to compare the performance of the optimal policy to that of the best static policy in order to characterize the performance improvement achievable through worker cross-training and dynamic line balancing. The MDP model is solved numerically, using a value iteration algorithm (see pp. 209–211 of [9]) with an accuracy of $\epsilon = 10^{-8}$. For most cases discussed in this paper, the computation time was not a concern. However, although the system state is quite efficient, it is not immune to the curse of dimensionality. For systems with $M \geq 12$ jobs in the system, computation times were long enough to make the solution of the model difficult.

3 Heuristics for Control of Worksharing

In Section 2, we argued that the true optimal policy is too complex to implement in practice. Therefore, heuristics with a simple structure such as a threshold policy are needed. In this section, we present an adaptation of the heuristic proposed by [5] that reflects our slightly different assumptions and another heuristic that we have developed.

Recall the SPT/RSPT policy of [6] presented in Section 1: the upstream worker transfers a unit to the downstream buffer if the number of "long" jobs (jobs whose shared task haven't been completed) exceeds half of the buffer size. The key idea in policies that aim to keep buffers half-full is the notion that the buffer should provide enough work for the downstream worker not to starve, and enough empty space for the upstream worker not to be blocked. [5] improved on the heuristic presented in [6] by dividing the tasks into identically distributed subtasks and counting the number

of all subtasks in the downstream buffer and station, instead of just counting the number of long jobs. Their rule states that the upstream worker will do the shared task if R or more units of work are present at the downstream station, where units of work are measured by the number of subtasks to be performed on the jobs waiting for service and the job being processed at the downstream station. Although the policy considered is slightly different, the assumptions made in [5] are very similar to those made in [6]; in both studies, it is assumed that once the shared portion of the work is started at the upstream station it has to be completed there. Thus, there is no preemption of the shared task. One difference between these two studies in terms of their assumptions is that [6] considered systems in which workers are allowed to pick the tasks with the shortest processing time while [5] devoted most of their attention to systems with FCFS job processing discipline, as we do in this paper.

McClain *et al.* demonstrated that using more accurate information on the content of the in-between buffer yields a more efficient half-full buffer heuristic, especially for systems using the FCFS rule. The following formula to calculate the cutoff level is an adaptation of the formula proposed in [5]. Our *half-full buffer* (HFB) heuristic is an enhancement of the original policy in the sense that it accounts for general mean processing times as well as different worker speeds when calculating the cutoff level. As explained in [5], the second term is the average work content in a half-full buffer whereas the first term represents the amount of work that will equalize the expected idling times of W1 and W2 (due to blocking and starvation, respectively) for a system with no buffers. Our adaptation results in a cutoff level of

$$R = \frac{T_{\mathsf{B}}}{1+v_2} + \frac{[((T_{\mathsf{B}}/v_2)+T_{\mathsf{C}})+T_{\mathsf{C}}]}{2}\frac{(M-2)}{2} \ . \tag{3.1}$$

In addition to the HFB heuristic, we have developed and tested another heuristic that we call the *50-50 work content* heuristic. This policy works as follows: W1 transfers the shared task to the downstream buffer if the ratio of the work content at station 1 to the total work content in the system is greater than 50% and starts processing it otherwise. We compute the anticipated amount of mean work content that would be in front of each worker if the shared task were to be transferred to the downstream station and "correct" the work content of W1 by a long-run balance statistic $y \in [0,1]$, which represents the fraction of jobs that must be processed in the B phase by W1 to balance the mean workloads of the workers. It is calculated as

$$y = \begin{cases} 0 & \text{if } \bar{y} < 0 \\ \bar{y} & \text{if } 0 \le \bar{y} \le 1 \\ 1 & \text{if } \bar{y} > 1 \end{cases} \tag{3.2}$$

where $\bar{y}$ is the solution to the equation

$$T_{\mathsf{A}} + y * T_{\mathsf{B}} = (1-y) * \frac{T_{\mathsf{B}}}{v_2} + T_{\mathsf{C}} \ . \tag{3.3}$$

Using the y factor provides a workload estimate that anticipates that some of the B tasks W1's buffer will be processed by W2 due to future transfers. Another point to note in the 50-50 work content heuristic is that the work contents are calculated assuming that the current B task at station 1 is transferred to buffer 2. This method gives a snapshot estimate of how the mean workload in the system would be distributed if W1 were to transfer his/her current task to the downstream station. For example, the mean work content ratio at state $(BOOCBC)$ for a system with $M = 5$ jobs is calculated as: $(1 * T_{\mathsf{A}} + 1 * y * T_{\mathsf{B}})/(1 * T_{\mathsf{A}} + 1 * y * T_{\mathsf{B}} + 2 * (T_{\mathsf{B}}/v_2) + 4 * T_{\mathsf{C}})$. This method of calculating the mean work content ratio inflates W2's workload and deflates W1's workload. Hence, unlike the HFB heuristic, this heuristic will give more of an incentive to *keep* the B task, as we will see more clearly in the test cases.

We use the measure of percent suboptimality to evaluate the performance of a heuristic. Percent suboptimality of a heuristic H is denoted by Δ^{H} and is expressed as

$$\Delta^{\mathsf{H}} = \frac{TH^* - TH^{\mathsf{H}}}{TH^*} 100\%$$

where TH^* and TH^{H} denote the throughput rates of the optimal and heuristic policies. We note that percent suboptimality is analogous to but distinct from the efficiency measure of [6], which measures the ratio of the throughput of a heuristic policy to an upper bound on the throughput calculated by setting worker utilizations to 100%.

To test our heuristics across a wide range of possible scenarios on the relative sizes of A, B and C tasks we set $M = 5$ and generated a set of 25 distinct test cases using permutations of 1.00, 1.50 and 2.00 for the values of μ_{A}, μ_{B} and μ_{C}. The average and maximum percent suboptimality of the 50-50 work content heuristic were 0.61% and 2.10%, whereas those for the HFB heuristic were 1.14% and 3.98%. We observed that for all cases in which the C task was longer than the A task the 50-50 work content heuristic performed better than the HFB heuristic and for all others (cases with A longer than C task) the HFB heuristic was the better heuristic. For cases with $T_{\mathsf{A}} > T_{\mathsf{C}}$, W1's fixed task (task A) is a bottleneck and the 50-50 work content heuristic which provides more incentive to *keep* utilizes more of W1's time for processing B tasks in comparison to the optimal policy and HFB heuristic and hence causes an ineffective use of W1's time. A similar argument is valid for the HFB heuristic in the $T_{\mathsf{A}} < T_{\mathsf{C}}$ case. When the two tasks are balanced ($T_{\mathsf{A}} = T_{\mathsf{C}}$) there is no clear bottleneck in the system and the 50-50 work content heuristic performs slightly better. Since the A and C tasks are equally likely to be long, it is intuitive that a policy that provides

more incentive to keep a job at the upstream station would perform better due to the constraint that prevents a transferred B task from returning back to the upstream station.

In addition to the test cases above, we tested how the heuristics perform at various WIP levels. Figure 1 plots the percent suboptimality versus WIP level for the HFB and 50-50 work content heuristics. The workers are assumed to be identical, and the mean processing times of the tasks are taken to be $T_{\mathsf{A}} = T_{\mathsf{C}} = 1/3$ and $T_{\mathsf{B}} = 1/2$. We selected a balanced case because the performance of the two heuristics were closest to each other for the balanced cases among the test cases we discussed above.

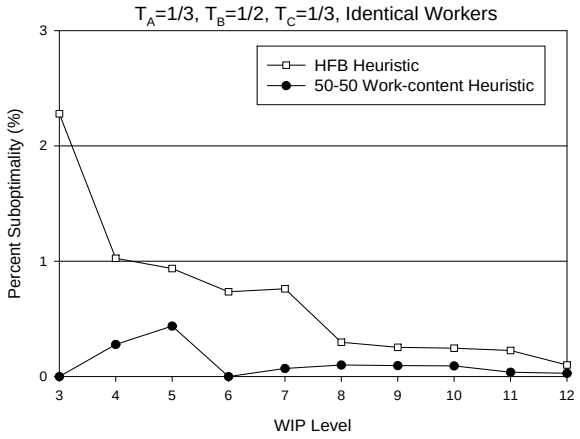


Figure 1: Percent suboptimality of HFB and 50-50 work content heuristics as a function of WIP Level

Figure 1 shows that the percent suboptimality of both heuristics go to zero as WIP level increases. As mentioned before, this can be attributed to the buffering capacity of increased WIP. However, even for low WIP levels we see that both heuristics perform very well with less than 2.5% suboptimality.

We observe in Figure 1 that the performance of the 50-50 work content heuristic is better than that of the HFB heuristic for all WIP levels, with less than 0.44% suboptimality. However, note that [5] assumed that the shared task is not preemptable, in contrast to our assumption up to this point. To examine the effect of our assumption, we set the number of jobs to $M = 5$ and looked at the suboptimality of the two heuristics at varying levels of flexibility as measured by the size of the shared task under both assumptions of preemptive and nonpreemptive shared task. We can make the following observations from Figure 2, which plots the percent suboptimality versus β for systems with identical workers and equal fixed tasks with $T_{\mathsf{A}} = T_{\mathsf{C}} = (10 - T_{\mathsf{B}})/2$:

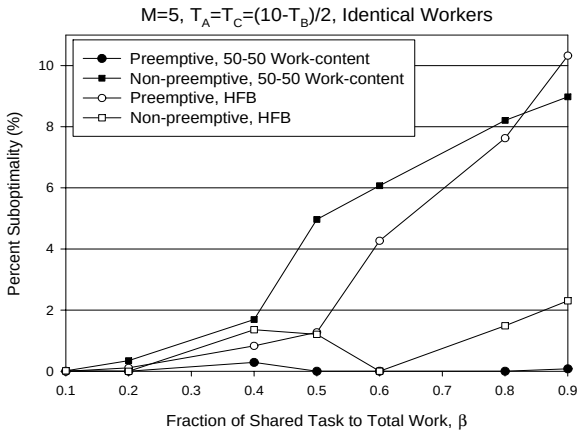


Figure 2: Percent suboptimality of HFB and 50-50 work content heuristics for preemptive and nonpreemptive systems

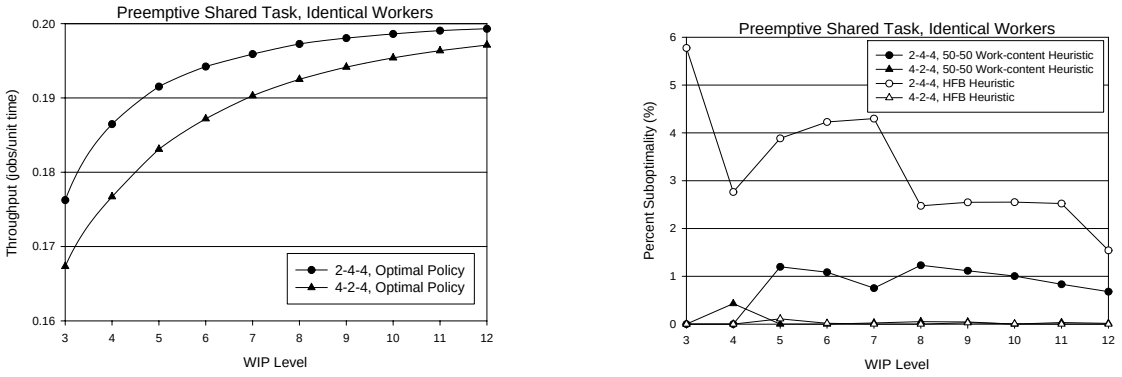
- If the system is preemptive, the 50-50 work content policy performs consistently well for all β levels, while the performance of the HFB heuristic deteriorates severely for $\beta > 0.5$ (i.e., when the shared task B is a distinct bottleneck).

- For a nonpreemptive system the performance of both heuristics deteriorates as β increases. However, the HFB heuristic consistently outperforms the 50-50 work content heuristic under the assumption that workers must complete tasks they have started.

The explanation for this observed behavior is, again, the relative incentive each heuristic puts on continuing to process the shared task at the upstream station versus transferring the shared task to the downstream station. Under the preemptive shared task assumption, the system is not really symmetric with respect to the two workers. A transferred shared task cannot return to the upstream station and therefore must be processed to completion by W2, whereas a non-transferred shared task can be processed by either W1 or W2. Hence, in a preemptive system, the control policy should put an increased incentive on keeping the shared task at the upstream station since it can always be transferred to the downstream station later. On the other hand, a nonpreemptive system possesses a greater symmetry in the sense that both workers must complete the tasks they have started, and hence the incentive to keep the shared task at the upstream station is greatly reduced.

We next explore the effect of unequal fixed tasks by comparing the 4-2-4 system ($T_{\mathsf{A}} = T_{\mathsf{C}} = 4$, $T_{\mathsf{B}} = 2$) with the 2-4-4 system ($T_{\mathsf{A}} = 2$, $T_{\mathsf{B}} = T_{\mathsf{C}} = 4$) under the assumption of a preemptive

shared task and identical worker speeds. Hence, both cases have the same total work content. Since $T_A + T_B + T_C = 10$ for both systems and all tasks are exponential, total variability in both systems is the same. So, any difference in performance is due to workload allocation. Figure 3(a) shows that the optimal throughput of the 2-4-4 system is higher than that of the 4-2-4 system for all WIP levels. This is intuitive, since we expect the performance to improve with increased flexibility up to moderate levels of β as discussed in the previous section.



(a) Optimal throughput versus WIP level

(b) Percent suboptimality versus WIP level

Figure 3: Percent suboptimality of HFB and 50-50 work content heuristics for equal and unequal fixed tasks

Figure 3(b) plots the suboptimality of the HFB and 50-50 work content heuristics. For the 4-2-4 system, both heuristics exhibit good performance with close to 0% suboptimality, but the performance of both heuristics deteriorates for the asymmetric 2-4-4 system. The superior performance of the 50-50 work content heuristic in the 2-4-4 system can be explained by the fact that $T_A < T_C$ and for such cases the 50-50 work content heuristic tends to perform better for the reasons discussed above. This observation suggests that performance of a heuristic is more robust in a symmetric system, and hence balancing the fixed tasks of the workers may be a more practical design choice where possible.

Finally, we examine the effect of different worker speeds on the performance of the heuristics. Several studies in the literature have reported that there may be significant differences in the processing speeds of the workers, particularly in industries where the turnover rate among employees is high [1]. Figure 4 plots the percent suboptimality of both heuristics versus β for three different values of v_2 under the assumption of a preemptive shared task. Although no uniform trend is

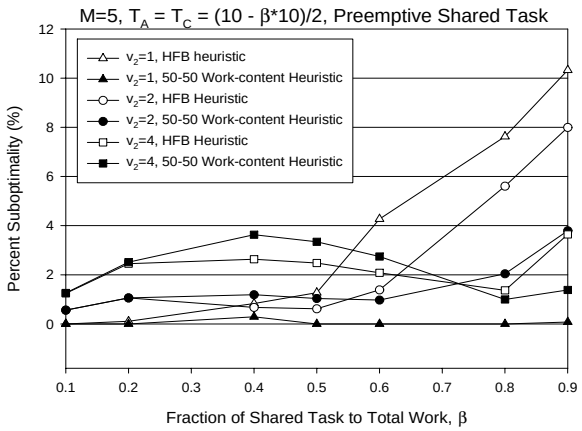


Figure 4: Percent suboptimality of HFB and 50-50 work content heuristics for different worker speeds

observable, we see that for low to moderate values of β, the performance of the 50-50 work content heuristic worsens as the downstream worker gets faster on the shared task ($v_2 = 2$, $v_2 = 4$), since the upstream worker becomes more of a bottleneck and therefore, the incentive to *keep* does not help. At high levels of β, the performance of the HFB heuristic deteriorates severely, although this trend is less observable for the $v_2 = 4$ case. The reason is that as W2 gets faster on the shared task, a "wrong" decision (transferring when he/she shouldn't) does not lead to as much ineffective utilization of the downstream worker. The same effect is responsible for the poor performance of the HFB heuristic at high β levels.

In summary, we conclude that the 50-50 work content policy performs extremely well in preemptive shared task systems with identical workers, and is quite robust to different worker speeds and unequal fixed tasks across a wide range of β values. However, if the system is nonpreemptive, using a heuristic that provides more incentive to transfer the task, such as the HFB heuristic, works much better than the 50-50 work content heuristic.

4 Two forms of buffering: WIP Inventory versus Worker Flexibility

A fundamental motivation for cross-training workers is to drive out idle time which occurs due to variability and/or a lack of balance in station capacities. Hence, we can think of worker flexibility as a form of buffer capacity that can be used in addition to or in place of WIP inventory and/or extra production capacity (i.e., more workers, faster processes). It is clear that this capacity can produce improvement in the performance of a production system in the form of increased throughput,

decreased WIP, or a combination of both. However, a key question that needs to be addressed is how much improvement is possible through the use of cross-training.

For the system described above, a measure of worker flexibility is the relative size of the shared task B compared to the total work (A+B+C). Following the notation in [6], we denote the fraction of the total work shared between the two workers as β. Under the assumption of identical worker speeds ($v_2 = 1$), β is calculated as $\beta = T_{\mathsf{B}}/(T_{\mathsf{A}} + T_{\mathsf{B}} + T_{\mathsf{C}})$. The system with $\beta = 0$ is a static system in which W1 and W2 process only fixed tasks.

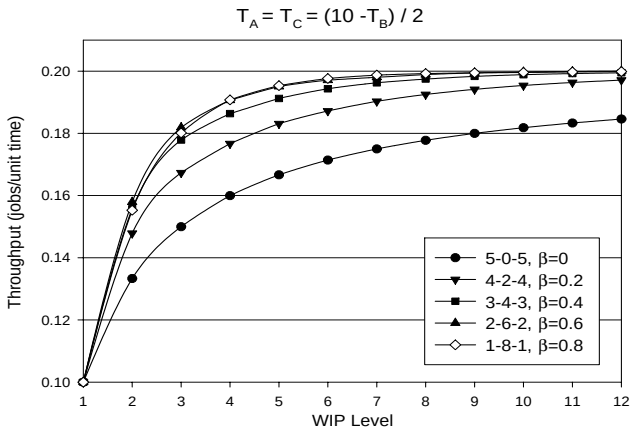


Figure 5: Optimal Throughput versus WIP Level (SCV of individual tasks A, B and C are fixed

We observe the performance of the optimal worksharing policy under various levels of flexibility as measured by β. The processing times for the fixed tasks A and C are assumed to be identically distributed exponential random variables to make the workers symmetric in terms of their assigned workloads. In the test cases we used, we set the total mean processing time, $T_{\mathsf{A}} + T_{\mathsf{B}} + T_{\mathsf{C}} = 10$ time units and computed the optimal policy and performance for $\beta = 0, 0.2, 0.4, 0.6$ and 0.8. For each case, the mean processing times for the tasks A, B and C were set by $T_{\mathsf{A}} = T_{\mathsf{C}} = 5(1 - \beta)$, $T_{\mathsf{B}} = 10\beta$. Figure 5 plots the throughput for the optimal worksharing policy as a function of WIP. We summarize our observations as follows.

- As expected, the optimal throughput of all systems approach the upper bound of 0.20 jobs per unit time as the WIP level increases since variability in the system eventually gets buffered by inventory. The benefits of cross-training are much more visible at low WIP levels since in the absence of inventory, systems with more flexibility achieve better performance through dynamic line balancing. For example, at a WIP level of ($M = 5$), system 4-2-4 performs

considerably better than system 5-0-5, achieving about 10% higher throughput.

- Benefits from cross-training can be realized as a throughput increase, WIP decrease, or a combination of the two. For example, system 4-2-4 achieves a throughput of 0.18 jobs per unit time with a WIP level of 5 jobs, whereas system 5-0-5 needs at least 9 jobs in the system to achieve the same throughput.

- There are diminishing returns to increased cross-training. The biggest performance improvement occurs from $\beta = 0$ to $\beta = 0.2$ and as β is increased beyond this level, the rate of improvement in system performance diminishes. This observation is an important one for real-life applications because it implies that partial cross-training implemented with an effective policy can achieve most of the logistical benefits achievable with full cross-training, which is almost always more expensive than partial cross-training.

As a result of the exponential processing times assumption, increasing the shared portion of the work while keeping the total mean processing time constant results in increased variability of the total processing time, which can cause anomalous behavior. Consider for example, the 2-6-2 and 1-8-1 systems, with coefficient of variations 0.66 and 0.81 respectively. For WIP levels less than four, the 1-8-1 system performs slightly worse than the less flexible 2-6-2 system, which can be explained by the high variability of the 1-8-1 system compared to that of the 2-6-2 system.

To examine the effect of increased flexibility while keeping the variability of the total work constant, we model task B as a combination of four distinct subtasks, B_1, B_2, B_3, and B_4, all of whose processing times are modeled as independent and identically distributed exponential random variables. We take $T_A = T_C = 1.0$ and $T_{B_1} = T_{B_2} = T_{B_3} = T_{B_4} = 2.0$, so that the expected processing time for the total work will be 10.0 time units. The SCV of the total processing time is equal to 0.42, which is a low variability case.

In this model, a system with $\beta = 0$ corresponds to assigning W1 to do tasks A, B_1, B_2 and W2 to do tasks B_3, B_4 and C. Note that we use this case as the model of specialized workers (i.e., no cross-training), which serves as a baseline for comparison with cross-trained workers. In the $\beta = 0.4$ system, tasks B_2 and B_3 are shared, while in the $\beta = 0.8$ system, both workers are cross-trained to perform all subtasks of B. Figure 6 plots the optimal throughput versus WIP for these three flexibility levels. Note that the behavior in Figure 6 is very similar to that in Figure 5: for a given WIP level the optimal throughput increases as the level of flexibility increases. The optimal throughput of the $\beta = 0.8$ system gets very close to the upper bound, even with only 3 jobs in the system.

Comparison of Figure 6 with Figure 5 reveals that reduced processing time variability has a

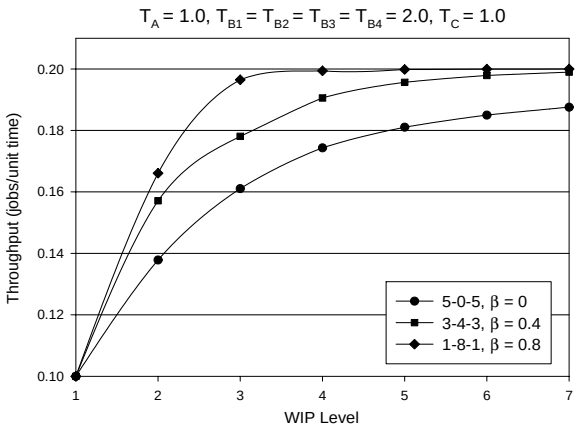


Figure 6: Optimal Throughput versus WIP Level (SCV of total work, A+B+C, is fixed)

significant impact on performance for $\beta = 0$ and $\beta = 0.8$, whereas it does not provide much change in performance for $\beta = 0.4$. For example, at WIP=3, the optimal throughput of the $\beta = 0.4$ system stays a little below 0.18 jobs per unit time for both settings assumed for Figure 6 and Figure 5. This observation suggests that moderate levels of β provides more robustness against increased variability. This is intuitive, since a long and highly variable shared task may cause W2 to spend a significant amount of time on the shared task, which eventually may lead to starvation of W1.

To summarize, we find that optimal system performance generally improves as the amount of cross-training or flexibility increases up to a certain level. Beyond that certain level, the effect of increased flexibility depends on the specific system characteristics, including the degree of variability. [6] reported severely declining efficiency for their SPT/RSPT rule at very high and low values of β under slightly different assumptions on the operation of the system. Unfortunately, the authors did not provide any insights into their finding. Our work suggests that the impact of variability may partially explain their result. But, the broader questions of how to quantify the value of cross-training and the conditions under which it is most beneficial still remain. In the next section, we discuss the extent to which our assumptions, specifically the assumption of preemption, affect the above findings.

5 System Architecture Factors Affecting Opportunity

In this section, we examine three factors that affect the performance of a worksharing policy: preemptability, granularity and variability. We provide insights and examine the effects of these factors in terms of opportunity, which we define here as the percentage improvement in throughput

rate achievable through worksharing relative to the throughput of the best static policy.

The definition of the best static policy for a given system depends on the specific assumptions. For example, if the shared task is preemptable it must be possible to divide the B task and statically assign a certain portion to W1 and the remainder to W2. So, for equal fixed tasks and a preemptive shared task, the system can be perfectly balanced under a static policy and hence, we use the performance of a balanced static policy as our standard for comparison for this case.

Comparing the performance of an optimal worksharing policy to the best static policy gives us an estimate for the logistical performance improvement that cross-training can bring to a manufacturing system. As such, it provides a measure that can be used to justify costs of cross-training. Formally, we define opportunity (Θ) as

$$\Theta = \frac{TH^* - TH^{\mathsf{St}}}{TH^{\mathsf{St}}} 100\ \% \ , \tag{5.1}$$

where TH^* and TH^{St} denote the throughput rate of the optimal worksharing and best static policies, respectively.

One factor affecting opportunity is WIP level, as we discussed in Section 4. As WIP increases opportunity decreases, since the performance of the static policy becomes closer to the performance of the optimal worksharing policy. This is due to increased variability buffering by WIP inventory, which enables higher utilization of both workers as a result of decreased starvation and blocking. In this section, we fix WIP at $M = 5$ to represent a reasonably low WIP level, which is where worksharing is most helpful.

Table 1 lists the test cases we used to examine the effects of preemption, granularity and variability. Case Set I has equal fixed tasks, while Case Set II has $T_{\mathsf{A}} + T_{\mathsf{B}} = T_{\mathsf{C}}$ so that allocating the shared task to W1 results in a balanced system. Hence, the best static policy is balanced for Case Set II, even if preemption is not allowed. In addition to the parameters of the cases, Table 1 provides the squared coefficient of variation (SCV) of the total job processing time (A+B+C) for each case under the assumption that processing time for the B task is distributed according to the exponential, Erlang-2, Erlang-3, or Erlang-4 distributions. Throughout this section, we assume that the workers are identical (i.e., $v_2 = 1$).

5.1 Preemption

The model described in Section 2 assumes that the shared task B can be transferred to the downstream buffer at any point of processing. That is, preemption is allowed. Several production environments have been suggested as examples in which this type of preemption is possible without significant time loss. One such example is the manufacturing of many types of sewn products such

		T_{A}	T_{B}	T_{C}	β	SCV of $T_{\mathsf{A}}+T_{\mathsf{B}}+T_{\mathsf{C}}$ for B $\sim$			
						Expon.	Erlang-2	Erlang-3	Erlang-4
Case Set 1	1	4.50	1.00	4.50	0.10	0.415	0.410	0.408	0.408
	2	4.00	2.00	4.00	0.20	0.360	0.340	0.333	0.330
	3	3.00	4.00	3.00	0.40	0.340	0.260	0.233	0.220
	4	2.50	5.00	2.50	0.50	0.375	0.250	0.208	0.188
	5	2.00	6.00	2.00	0.60	0.440	0.260	0.200	0.170
	6	1.00	8.00	1.00	0.80	0.660	0.340	0.233	0.180
	7	0.50	9.00	0.50	0.90	0.815	0.410	0.275	0.208
Case Set 2	1	4.50	0.50	5.00	0.05	0.455	0.454	0.453	0.453
	2	4.00	1.00	5.00	0.10	0.420	0.415	0.413	0.413
	3	3.50	1.50	5.00	0.15	0.395	0.384	0.380	0.378
	4	3.00	2.00	5.00	0.20	0.380	0.360	0.353	0.350
	5	2.50	2.50	5.00	0.25	0.375	0.344	0.333	0.328
	6	2.00	3.00	5.00	0.30	0.380	0.335	0.320	0.313
	7	1.50	3.50	5.00	0.35	0.395	0.334	0.313	0.303
	8	1.00	4.00	5.00	0.40	0.420	0.340	0.313	0.300
	9	0.50	4.50	5.00	0.45	0.455	0.354	0.320	0.303

Table 1: Test cases and SCV of the total work for each model

as apparel, furniture, shoes, etc. [1], [3]. In these systems, a well established policy is the *Toyota Sewn Products Management System* (TSS). TSS and its variation, the bucket brigade policy, rely on the ability to preempt a task in process. The basic rule in TSS is that each worker processes a job on each station along the production line until he/she gets bumped by a downstream worker. In the apparel industry, the equipment (e.g., sewing machines) is general purpose and hence does not require extensive additional training to fully cross-train workers for different tasks. Furthermore, the nature of the operations is such that workers can take over another worker's job with little time loss or difficulty. Another example of this type of environment is order picking in warehouses, where workers can easily transfer jobs (orders). [2] implemented the bucket brigade policy in the warehouses of a major chain retailer and reported a more than 30% increase in pick rates without a conscious effort of workers. However, there exist many production environments in which it is impossible or costly to stop processing an item and transfer it to the downstream buffer once processing has started. Hence, it is important to examine the impact of this assumption on the opportunity for worksharing.

If the shared task is not preemptable, then starting a B task at station 1 means that it must be processed to completion there. Hence, the "regret" of a suboptimal decision is much higher in

a nonpreemptive system. In a preemptive system, each task completion is a decision epoch, which allows the system to evaluate and modify decisions more frequently and thus avoid consequences of a suboptimal decision. Since the set of decision epochs of a nonpreemptive system is a subset of that of a preemptive system, the optimal throughput for a preemptive system must always be at least as high as that for a system with a nonpreemptive assumption. However, the "best" static policy will also perform better under the preemptive shared task assumption, since the system will be more flexible in terms of how tasks are allocated to workers.

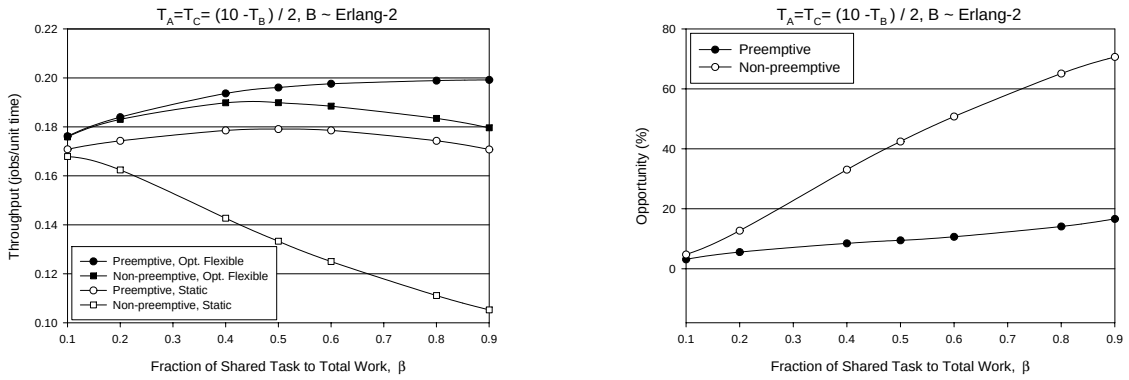
Consider a deterministic system with $T_{\mathsf{A}} = 2$, $T_{\mathsf{B}} = 4$ and $T_{\mathsf{C}} = 4$. When the B task is preemptive, we can allocate work such that each worker will process each job for 5 time units, resulting in a throughput of 1/5 jobs per unit time, which is the maximum throughput achievable. Clearly, as long as it is possible to balance the workloads of the workers in a deterministic system, worker flexibility does not offer any opportunity. However, if the system is stochastic, worker flexibility can provide additional buffering which is not possible under a static allocation policy. Hence, flexibility offers opportunity in stochastic preemptive settings even if perfect balancing of the workloads is possible with a static policy. In a nonpreemptive system, however, the best throughput that a static policy could achieve is 1/6 jobs per unit time, whereas a throughput of 1/5 jobs per unit time is achievable if the B task is shared. Hence, worksharing has opportunity even for deterministic nonpreemptive systems due to capacity balancing.

We examine at the effect of preemption using both case sets of Table 1. Case Set I represents systems in which only the preemptive system can be perfectly balanced by dividing the shared task into two parts. In contrast, in Case Set II preemption does not help to balance the static policy since the best static policy allocates all of the shared task to W1. We model the shared task B as the sum of two exponential subtasks, B_1 and B_2, where $E[S_{\mathsf{B}_1}(\cdot)] = E[S_{\mathsf{B}_2}(\cdot)] = T_{\mathsf{B}}/2$. Hence, the total processing time for task B is distributed as an Erlang-2 random variable. This allows us to keep the total variability constant while comparing the optimal worksharing and static policies.

Figure 7(a) plots the throughput of the optimal worksharing and static policies as a function of β for Case Set I, for both preemptive and nonpreemptive shared task assumptions. The opportunity under these two assumptions is presented in Figure 7(b). We make the following observations about the performance of the optimal worksharing policy:

- The throughput of the static policy for the preemptive system (balanced static policy) is maximized when $\beta = 0.5$, since that is the lowest variability case ($T_{\mathsf{A}} = T_{\mathsf{B}_1} = T_{\mathsf{B}_2} = T_{\mathsf{C}} = 2.5$ with an SCV of 0.25).

- In contrast, the throughput of the static policy for the nonpreemptive system is strictly



(a) Throughput versus β

(b) Opportunity versus β

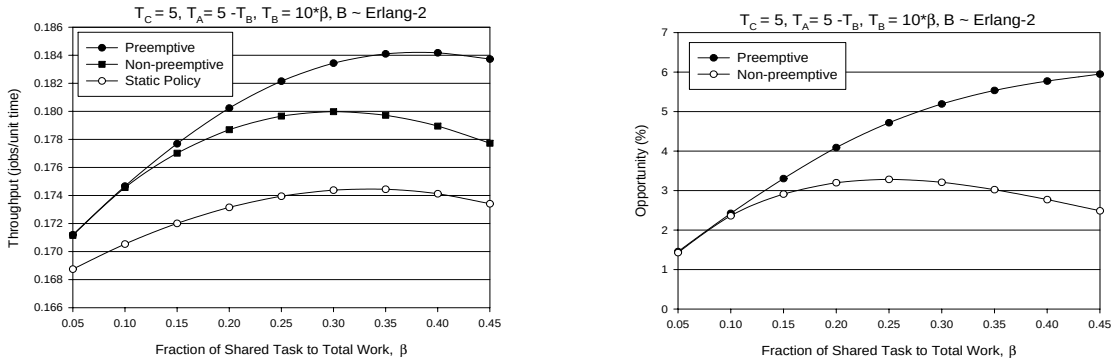
Figure 7: Opportunity for preemptive and nonpreemptive systems (Case Set I)

decreasing as β increases, due to the fact that the workloads of the workers become more unbalanced.

- The optimal throughput of the preemptive system increases as β increases, approaching the upper bound of 0.20, as expected.

- The optimal throughput of the nonpreemptive system follows that of the preemptive system closely up to a certain level of $\beta \leq 0.4$. However, for $\beta \geq 0.5$, we see that the performance of the nonpreemptive worksharing policy is strictly decreasing, while that of the preemptive worksharing policy is strictly increasing. For $\beta = 0.8$, for example, preemption improves the optimal throughput by about 8%.

As the length of the shared task increases, the penalty of a *put* or *keep* decision that turns out to be "wrong" along the sample path realization is magnified, which deteriorates the performance of a nonpreemptive system. To see this, suppose that preemption of the shared task from W1 is not allowed and at a decision epoch with state $(BOOOBC)$, W1 starts processing the B task. Since the B task has been started, it must be processed to completion at station 1. This means that if W2 completes his/her tasks before W1 completes the B task, W2 idles due to starvation. For systems with a long shared task, this effect can be very pronounced, which partially explains the last observation. However, we must also note that the total variability is also increasing for $\beta > 0.5$, which affects the nonpreemptive system more adversely than the preemptive system. We will examine the effects of variability in Section 5.3.



(a) Throughput versus β (b) Opportunity versus β

Figure 8: Opportunity for preemptive and nonpreemptive systems (Case Set II)

From Figure 7(b), we see that for both preemptive and nonpreemptive systems, opportunity increases as the amount of flexibility (as measured by β) increases. The opportunity for the non-preemptive system is higher than that for the preemptive system for all β levels. This is again due to the capacity balancing offered by worksharing, which might not be possible by static allocation policies under the nonpreemptive shared task assumption.

Figures 8(a) and 8(b) plot the throughput and opportunity versus β, respectively, for Case Set II, in which the static policy for both preemptive and nonpreemptive systems are balanced. Again, we see that opportunity for the preemptive system increases as β increases. This time, however, we see that the worksharing opportunity is very limited —less than 7% in contrast to the high values of opportunity we observed for Case Set I. Furthermore, the opportunity for the nonpreemptive system decreases for $\beta \geq 0.25$, which can be explained by the fact that task C is a severe bottleneck for all of these test cases ($T_C = 5$). As the shared task gets longer, helping the upstream worker uses more and more of the capacity that W2 should spend on processing C tasks. In fact, we see this effect in both preemptive and nonpreemptive systems; the optimal throughput of the preemptive system decreases for $\beta > 0.40$, which is an implication of the assumption that once a task is sent downstream, it cannot return. The nonpreemptive system is even more sensitive, because the preemptive system can avoid transferring the shared task to the downstream buffer unnecessarily, since it can always be transferred later on, whereas the nonpreemptive system has to make a one-time decision.

These examples illustrate that whether or not the shared task can be preempted while in

progress is an important determinant of how much improvement cross-training offers in a manufacturing environment. In principle, preemption enables effective sharing of work between the two workers and decreases the need for optimal control since the decision to *keep* a B task can be modified even after the processing is started. The opportunity, on the other hand, depends on whether worksharing achieves better capacity balancing compared to a static policy. In systems where worksharing only provides variability buffering, the opportunity will be limited, whereas unbalanced systems with discrete, non-preemptive tasks offer significant improvement opportunity through worker cross-training. However, in systems with rigid task definitions, we found that more cross-training (as measured in the fraction of shared task to total work) may not always result in better system performance, which was also observed in previous work on the subject [6]. Hence, determining an appropriate level of cross-training as well as an effective control policy, is an important part of a design effort for workforce agility.

5.2 Granularity

In the systems we have analyzed so far we have allowed the shared task to be either preemptive or nonpreemptive. That is, we have only considered systems in which the decision to *keep* or *put* can be made either at any point throughout the processing of the B task at the upstream station, or only before the B task processing starts. However, in between these extremes are the cases where there are natural break points in the shared task, so that the shared task can be transferred to the downstream station only at these points.

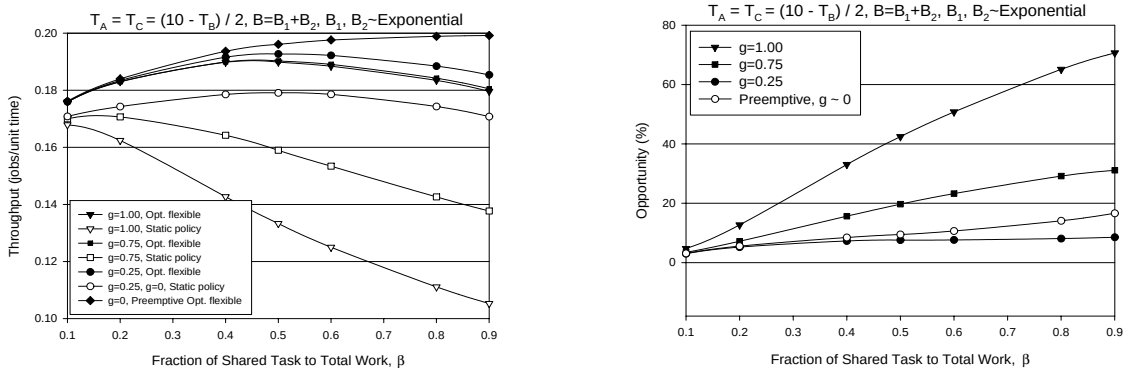
One real life example of such an environment we have observed is the casting finishing line of a steel foundry. The company produces railroad car coupler castings that weigh several hundred pounds and must be manually finished through operations like grinding high spots, welding holes and gauging dimensions. The processing of each side of a coupler requires about 7 distinct tasks which were originally grouped as a single task since rotation and handling of the coupler is cumbersome. Although preemption during the processing of a "subtask" such as welding is not practical, distinct subtasks provide natural break points at which another worker could take over the processing of the remaining subtasks on a side.

We define *granularity* as a measure that accounts for the number of break points and the size of subtasks that are transferable in the shared task. For example, the nonpreemptive system of the previous section considers a shared task of maximum granularity, whereas the preemptive system represents the limiting case of zero granularity. An appropriate metric for granularity, g, is

$$g = \frac{1}{\sum_{i=1}^{K} T_{B_i}} \prod_{i=1}^{K} T_{B_i} \ , \tag{5.2}$$

where T_{B_i} denotes the mean processing time of the ith subtask of the shared task. Note that g is a measure of how many distinct subtasks the shared task can be divided into, as well as the relative sizes of the subtasks. For example, if the shared task can only be transferred at the beginning of processing, then $g = 1.00$. In contrast, if the shared task has two equal subtasks, then $g = 0.25$. Note that this is lower than the granularity measure for the case where the shared task has one short subtask and one long subtask that is three times as long as the short subtask, which is $g = 0.75$.

To observe how granularity affects the performance and opportunity of worksharing, we use the test cases in Case Set I. For $g = 1.00$ and $g = 0.25$, we take the processing time of task B to be the sum of two independent identically distributed exponential random variables (i.e., $S_B \sim$ Erlang-2). This ensures that the processing time variability for all test cases is equal for both granularity levels and hence eliminates any variability effects. For $g = 0.75$, the distribution of the processing time for task B is taken to be the sum of two exponential random variables with means $T_B/4$ and $3T_B/4$, which results in a higher processing time variability for the shared task, B. To make a fair comparison, the $g = 0.75$ case should be compared to a $g = 1.00$ system in which task B consists of exponential subtasks with means $T_B/4$ and $3T_B/4$. However, we note that this $g = 1.00$ system represents a higher variability case than the $g = 1.00$ system with Erlang-2 distributed shared task and hence, has worse performance than the latter. Thus, a comparison of the $g = 0.75$ case with the $g = 1.00$ system with Erlang-2 distributed shared task is meaningful.



(a) Throughput versus β

(b) Opportunity versus β

Figure 9: Effect of granularity on opportunity (Case Set I)

Figures 9(a) and 9(b) show the optimal throughput and opportunity of worksharing for Case Set I under different levels of granularity ($g = 1.00$, $g = 0.75$ and $g = 0.25$). The performance of the best

static policy is also provided for comparison. Since the fixed tasks of W1 and W2 are taken to be equal in Case Set I, the static policy is a perfectly balanced system for $g = 0.25$. When $g = 0.75$, the best static policy allocates 1/4 of task B to W1 and 3/4 of it to W2 in addition to their fixed tasks, which results in an unbalanced system. The following points summarize our observations.

- For all β levels, throughput of the optimal worksharing policy increases as granularity decreases.
- A less granular shared task results in a more balanced static system, so performance of the static policy also improves as granularity decreases.
- Among the nonpreemptive systems, opportunity increases as granularity increases. However, we also observe that the opportunity of the preemptive system is higher than that of the $g = 0.25$ system.

The last observation is a consequence of the fact that the best static policy is perfectly balanced for both the $g = 0.25$ system and the preemptive system. Hence, which system has higher opportunity depends on which worksharing system has better optimal performance. The preemptive system represents a limiting case of granularity ($g \to 0$). Since the set of decision epochs in a system with $g > 0$ will be a subset of those in a preemptive system along any sample path, the optimal performance of a preemptive system is always at least as good as a $g > 0$ system, all other things being equal.

As expected, we found that the performance of the optimal worksharing policy improves as the granularity of the shared task decreases. Our analysis shows that in nonpreemptive systems where significant opportunity exists due to the capacity balancing effect of worksharing, a way to further improve the performance is to introduce natural break points in the processing of shared tasks. This observation has a direct application for the finishing line of the steel foundry we studied. The operations on a single side of a coupler should be treated as distinct subtasks with worksharing decisions made at subtask boundaries rather than considering an entire side as indivisible. This would decrease the granularity of the shared tasks and as a result, would improve the performance of a team of cross-trained workers.

5.3 Variability

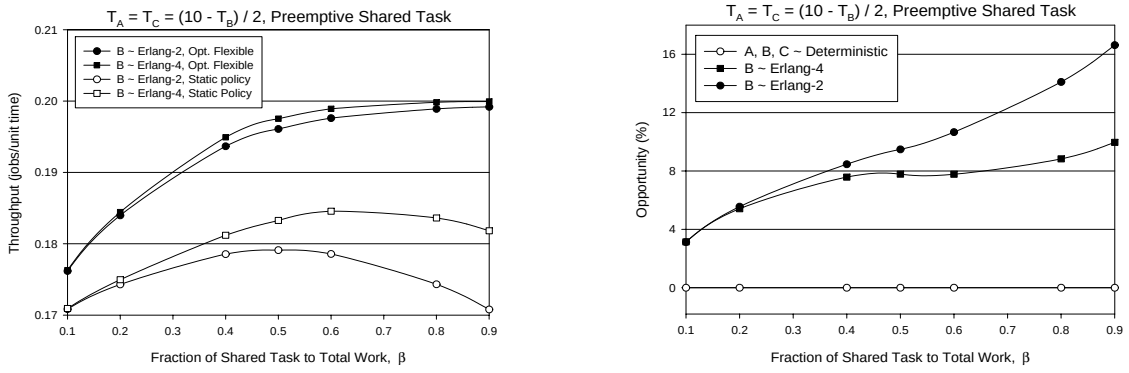
In this section, we examine the effect of variability on the opportunity of worksharing. As the variability of the shared task (and hence that of the total processing time) increases, we expect the performance of any policy, flexible or static, to deteriorate. As we will show, the qualitative effect

of variability on opportunity depends on how well the optimal worksharing policy accommodates increased variability compared to the best static policy.

As discussed before, there are two performance improving effects that the optimal worksharing policy provides: capacity balancing and variability buffering. Worksharing provides variability buffering in all the systems we have discussed so far. For example, if an A task is unexpectedly long, the worksharing policy may dictate that the B task be transferred downstream, since during the processing of that long A task the queue in front of W2 may have been depleted. In a static policy, however, it is not possible to buffer against unexpected events in this sense.

For nonpreemptive systems with equal fixed tasks ($T_{\mathsf{A}} = T_{\mathsf{C}}$), we assumed that the best a static policy can do is to allocate the shared task to either one of the workers. As Figure 7(b) shows, the opportunity in this case can get as large as 70% because the opportunity of worksharing is also due to capacity balancing. Compared to the variability buffering effect, capacity balancing effect provided by worksharing increases opportunity drastically.

To examine the effect of increased processing time variability on opportunity, we first look at the opportunity in a preemptive system with equal fixed tasks (Case Set I), where worksharing only provides variability buffering. We model the shared task processing time as the sum of identically distributed exponential variables to look at different levels of variability. Using Erlang-2 and Erlang-4 distributions for the shared task processing time proves appropriate since the static policy allocates half of the shared task to W1 and the other half to W2 and hence the variability remains the same for the static policy. Figures 10(a) and 10(b) plot the throughput and opportunity as a function of



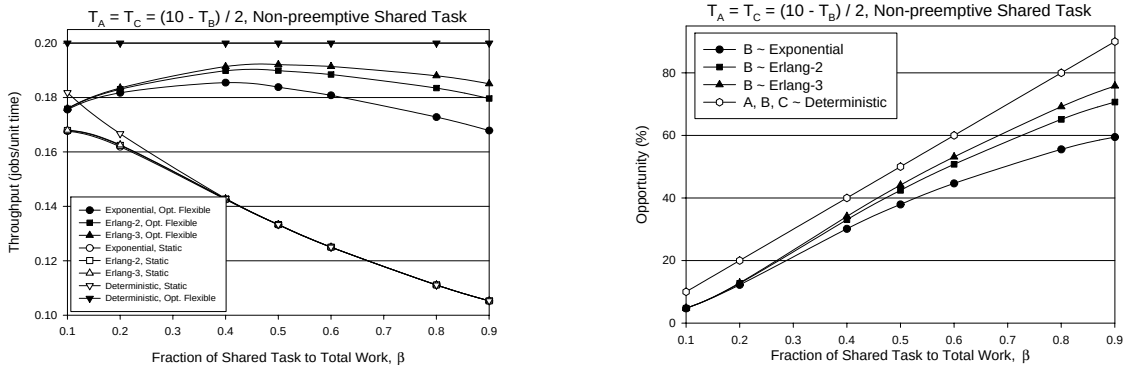
(a) Throughput versus β (b) Opportunity versus β

Figure 10: Effect of variability in a preemptive system (Case Set I)

β for the two variability levels as well as deterministic processing times for tasks A, B and C. The opportunity for the deterministic case is always zero, since the static policy achieves a perfectly balanced system in this case and will achieve the same throughput as an optimal flexible policy.

As expected, increased variability results in decreased throughput for the optimal worksharing policy as well as the best static policy. However, we see that the deterioration for the optimal flexible policy is much less than that for the best static policy. In other words, the optimal worksharing policy is much more robust to increased variability in processing times. This causes the opportunity of the less variable system (i.e. system with Erlang-4 shared task) to be lower for all β levels.

We next demonstrate that the effect of variability may be reversed in a system in which worksharing provides capacity balancing in addition to variability buffering. For this purpose, we examine the opportunity for Case Set I under the assumption that the shared task is nonpreemptive. Recall that the fixed tasks in Case Set I are equal and since the shared task is nonpreemptive, we assume that the best static policy allocates the shared task to either of the workers and hence the capacities of the workers can get severely unbalanced for high values of β. Figures 11(a) and 11(b) show the throughput and opportunity for Case Set I ($T_A = T_C = 5(1-\beta)$) under three distributions (exponential, Erlang-2, Erlang-3) for the shared task processing time as well as the deterministic processing times case. In the deterministic case, the opportunity of worksharing increases linearly



(a) Throughput versus β (b) Opportunity versus β

Figure 11: Effect of variability in a nonpreemptive system (Case Set I)

as a function of β; the throughput of the best static policy is equal to $1/[5(1+\beta)]$ and the optimal flexible policy achieves the maximum throughput of $1/5$ for all β levels, which yields $\Theta = 100\beta\%$.

The performance of the optimal worksharing policy improves as variability decreases. However,

the performance of the static policy is not significantly affected by the variability, since for all β levels the static policy yields an unbalanced system in which the performance is largely determined by this first order effect. Compared to the preemptive shared task case, the effect of variability on the performance of the optimal worksharing policy is more visible in the nonpreemptive system which is intuitive since a preemptive system can modify actions as information on the processing time of the B task becomes available.

6 Conclusions

In this paper, we examined dynamic line balancing in two-station systems operating under a CONWIP job release policy. Specifically, we explored three system architecture factors that affect the efficacy of a worksharing policy: preemption, granularity and variability of the shared task. We did this in the context of the optimal worksharing policy (computed using an MDP model) and also heuristic policies, which are more practical from an implementation standpoint. We evaluated two threshold type heuristics: the half-full buffer (HFB) heuristic and the 50-50 work content heuristic, both of which are simple, implementable policies that aim to balance the workloads of the workers. Compared to the HFB heuristic, the 50-50 work content heuristic provides more of an incentive to keep the shared task at the upstream station and hence is more suitable to systems in which preemption of the shared task is possible. In most cases, however, the performance of both heuristics was satisfactory. This is encouraging, since these rules are practical for real-world systems.

Our analyses showed that preemptability of the shared task is a strong determinant of both the opportunity offered by worksharing and the performance of control heuristics. In systems where the shared task is preemptable, more cross-training generally results in better system performance in terms of throughput, which is intuitive. However, in nonpreemptive systems, our results confirmed the perhaps counter-intuitive findings of [6] that cross-training beyond a certain point harms rather than helps system performance. Although we did not focus on finding the optimal amount of cross-training, our results suggest that the fraction of shared task to total work should be roughly between 30% to 50% in nonpreemptive systems. However, the optimal level depends on system characteristics, so this is an issue that deserves further research attention.

Although the determination of the right amount of cross-training and the design of an effective control policy present bigger challenges in a nonpreemptive system, the opportunity for such systems is also much larger than that for preemptive systems. Since the processing of a shared task can be stopped to be resumed later at the downstream station, a preemptive system implies that it is possible to balance the system through a new job design that allows a better division of work between workers. In such systems, the performance opportunity that dynamic line balancing offers

comes from variability buffering and not from capacity balancing, which restricts it to much lower levels (about 10-20%). In a nonpreemptive environment, on the other hand, worksharing can achieve long-term capacity balancing, which may not be possible through job design under a static policy. Hence, we believe that systems in which preemption of tasks is not practical are good candidates for workforce agility since even a low level of worker flexibility can improve the system performance considerably.

Increased opportunity in nonpreemptive systems can be achieved by the introduction of natural break points in the operation to decrease the granularity of the shared task. This enables more effective sharing of work between the two workers and makes the system more robust to variability in the system. This is an important issue since we have shown that the performance of a worksharing policy in a nonpreemptive system is much more sensitive to variability (i.e., unexpectedly long or short tasks) than in a preemptive system.

Finally, an important contribution of this study is the explicit demonstration of the trade-off between WIP and worker flexibility. Workforce agility can be used to achieve a combination of WIP decrease and throughput increase depending on the particular system's needs and objectives. Furthermore, extensive cross-training of the workers is not a necessity since partial cross-training, when organized carefully, can provide most of the logistical benefits. One shortcoming of our models is that they are restricted to two-station systems. However, consideration of two-station systems provides us with sufficiently simple analytical models that yield insights on how different factors in the physical environment affect the performance opportunity and interact with each other. We believe that these insights are valid for longer lines and even more complex routings. Further research however, is needed to evaluate the effectiveness of threshold policy heuristics in longer lines. Even though there is evidence on the effectiveness of the HFB heuristic in longer lines ([6], [5]), it remains to be seen whether a multi-station heuristic that uses the principles of the 50-50 work content heuristic offers greater opportunity in certain classes of systems.

References

[1] Bartholdi, J.J. and Eisenstein, D.D. (1996a) A production line that balances itself. *Operations Research*, 44(1), 21–34.

[2] Bartholdi, J.J. and Eisenstein, D.D. (1996b) A self-balancing order-picking system for a warehouse. Technical Report, Dept. of Industrial Engineering, Georgia Institute of Technology, Atlanta, GA.

[3] Bartholdi, J.J., Bunimovich, L.A., and Eisenstein, D.D. (1999) Dynamics of two- and three-worker "bucket brigade" production lines, *Operations Research*, 47(3), 488–491.

[4] Bertsekas, D.P. (1987)*Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, Englewood Cliffs.

[5] McClain, J.O., Thomas, L.J. and Sox, C. (1992) On-the-fly line balancing with very little WIP, *International Journal of Production Economics*, 27, 283–289.

[6] Ostolaza, J., McClain, J.O., and Thomas, L.J. (1990) The use of dynamic (state-dependent) assembly-line balancing to improve throughput, *Journal of Manufacturing Operations Management*,3, 105–133.

[7] Powell, S.G. and Schultz, K.L. (1998). Modeling and worker motivation in JIT production systems. Technical Report, Tuck School of Business Administration, Dartmouth College.

[8] Schultz, K.L., Juran, D.C., Boudreau, J.W., McClain, J.O., and Thomas, L.J. (1998). Modeling and worker motivation in JIT production systems. *Management Science*, 44(12), 1595–1607.

[9] Tijms, H.C. (1986). *Stochastic Modeling and Analysis: A Computational Approach.* Wiley, New York, NY.

[10] Van Oyen, M.P., Gel, E.S., and Hopp, W.J. (2000) Performance opportunity of workforce agility in collaborative and noncollaborative work systems, Forthcoming in *IIE Transactions* Technical Report (http://www.iems.nwu.edu/ vanoyen/).

[11] Zavadlav, E., McClain, J.O. and Thomas, L.J. (1996) Self-buffering, self-balancing, self-flushing production lines, *Management Science*, 42(8), 1151–1164.