Adv. Appl. Prob. **38,** 621–642 (2006)
Printed in Northern Ireland
© *Applied Probability Trust* 2006

DYNAMIC LOAD BALANCING WITH FLEXIBLE WORKERS

HYUN-SOO AHN,* *University of Michigan*
RHONDA RIGHTER,** *University of California, Berkeley*

Abstract

We study the problem of dynamically allocating flexible workers to stations in tandem or serial manufacturing systems. Workers are trained to do a subset of consecutive tasks. We show that the optimal policy is often LBFS (last buffer first-served) or FBFS (first buffer first-served). These results generalize earlier results on the optimality of the pick-and-run, expedite, and bucket brigade-type policies. We also show that, for exponential processing times and general manufacturing networks, the optimal policy will tend to have several workers assigned to the same station.

Keywords: Stochastic scheduling; tandem systems; multiserver queue; flexible worker; load balancing

2000 Mathematics Subject Classification: Primary 90B36
Secondary 90B22; 60K25

1. Introduction

Increasing the flexibility of workers through cross-training or work sharing, and effectively deploying flexible workers, has become more important as companies strive to reduce cycle times while maintaining high utilization. The use of flexible workers permits a manufacturing system to balance itself by dynamically shifting workloads (or, equivalently, shifting workers) in response to changes in demand, machine availability, congestion points, etc., even when there may be a large variation in worker speeds and task completion times. The self-balancing capability in turn results in lower inventory levels. Worker flexibility may also have positive effects on worker morale and ergonomics, quality, and system coordination. We develop a general framework for studying the dynamic assignment of flexible workers in open and closed tandem or serial manufacturing systems. Of course, for the most general models, optimal policies will be difficult to find and impractical to implement. We characterize particular models and objective functions for which easily implementable policies are optimal, at least within a subsystem of the general system. These policies provide partial characterizations of the optimal policy in more complicated systems, and can be used to develop heuristics.

Most of our results apply to tandem systems in which jobs consist of n tasks that must be done in sequence. Jobs ready and waiting for task j to be performed are said to be at station j, or in buffer j, where buffers have infinite capacity. We are able to give a few results for general independent processing times, as well as stronger results for deterministic and exponential processing times. Arrivals of jobs may be either independent of the system state but otherwise

Received 21 May 2004; revision received 11 April 2006.
* Postal address: Department of Operations and Management Science, Ross School of Business, University of Michigan, 701 Tappan Street, Ann Arbor, MI 48109-1234, USA. Email address: hsahn@umich.edu
** Postal address: Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720, USA. Email address: rrighter@ieor.berkeley.edu

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

arbitrary (an open system), or such that the completion of a job may trigger an immediate new job arrival (a closed, or CONWIP, system). There are w workers, or servers, and they are cross-trained, so that they can do multiple tasks and tasks can be done by more than one worker. We assume that each worker has a 'zone' of capability, or is sequentially cross-trained, so that worker i can do tasks $k_i, k_i + 1, \ldots, l_i$ for some k_i and l_i, and that k_i and l_i are increasing (nonstrictly) in i, with $k_1 = 1$ and $l_w = n$. If some worker i is the only one that can do a set of tasks, we call those tasks worker i's fixed tasks (cf. [15]). Tasks that can be done by more than one worker are called shared tasks. The zone approach to cross-training is practical for linear and U-shaped layouts, and is consistent with both the moving worker modules and the dynamic assembly-line balancing models of worksharing for tandem or assembly-line systems, as described in [6].

Moving worker module systems (see, e.g. [11]) have fixed machines for each task, and workers may move from one machine to another. In dynamic assembly-line balancing systems, introduced in [23], machines are generally assumed to be flexible, workers stay with machines, and some tasks may be done at more than one flexible machine. These systems are designed so that workers can easily help adjacent workers when they fall behind, by working on shared tasks. In dynamic assembly-line balancing systems there may be additional constraints on the flow of jobs. For example, it may be that, once a worker starts working on a job that was taken from an upstream worker, the job is considered 'handed off' to the former worker and later tasks for that job cannot be done by upstream workers. These constraints are not included in our model. We assume that switching times between tasks, or walking times between stations, are insignificant, as are hand-off times from one worker to another, and that preemption is generally permitted. If there are multiple jobs at a station, multiple workers can work on them. Most of our results assume a noncollaborative model, so that multiple workers cannot work on the same job, though we have a few results that apply when workers can collaborate on the same job, where their processing rates are added. Workers may have random failures (or go on breaks) and repairs (returns).

In many situations an FBFS (first buffer first-served) or LBFS (last buffer first-served) policy is optimal. By FBFS and LBFS we mean that every worker works on the job that is in the earliest or, respectively, latest buffer (or station) among the buffers it is qualified to serve, that service is FCFS (first-come–first-served) within each buffer, and that, when there is contention among workers for the same job, the worker that can do the corresponding task the fastest (on average) is given priority. Many policies in the literature can be recast as FBFS or LBFS policies. For example, for some task service time distributions, the shortest expected remaining processing time policy for jobs is equivalent to the LBFS policy in our tandem model. Also, in certain situations, LBFS is equivalent to the pick-and-run, expedite, and (a variant of the) bucket brigade policies described below.

Much of the research on dynamic worker assignment has focused on two-stage systems with only one flexible worker (or 'floater') and with holding and switching costs [12], [13], [16], [18], [27]. In these models the optimal policy typically has a monotone switching curve structure. Pandelis [24], [25] considered a two-stage system with both flexible and dedicated workers and also showed a switching curve structure for the optimal policy. Gel *et al.* [15] and Sennott *et al.* [27] considered both two- and three-stage systems with one flexible worker. Gel *et al.* showed conditions under which a 'fixed before shared' policy maximizes the long-run throughput. Under their assumptions, the fixed before shared policy corresponds to a LBFS policy; we give related results where the fixed before shared policy is optimal for two or more stages when processing times are deterministic or exponential.

There is also a sizable literature on two-stage systems in which all workers are fully trained, processing times depend on the task and not the worker, and there are no switchover costs or

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

times for workers to switch tasks or jobs to switch workers. Ahn *et al.* [2] considered a two-station model with two flexible workers, Poisson arrivals, exponential service times, holding costs, and preemption. They considered both the cases when the workers can collaborate and the cases when they cannot, and their objective was to minimize the long-run average cost. They showed that, in states where both workers can be assigned to either station, assigning both to the same station is always optimal. They gave conditions under which it is optimal to serve one of the stations exhaustively (i.e. either FBFS or LBFS). Ahn *et al.* [3] considered a similar nonpreemptive, nonidling model without arrivals. Pandelis and Teneketzis [26] studied a nonpreemptive, noncollaborative two-station model with multiple fully trained workers, no arrivals, and holding costs. Jobs that complete the first task require the second task with probability p; otherwise, they leave immediately. Pandelis and Teneketzis gave conditions under which the FBFS policy is optimal for general task-dependent service times. Javidi *et al.* [19] considered a similar model with Poisson arrivals and exponential services. Jobs that complete the first task create k jobs requiring the second task with probability p; otherwise, they leave immediately. Javidi *et al.* gave optimality conditions for FBFS and for LBFS where the objective is to minimize the time until the system first empties.

Andradóttir *et al.* [4] studied a finite-buffer system with fully trained workers that can collaborate on jobs and where service times are general and may depend on both the task and the worker. Their objective was to maximize the steady-state throughput. Preemption is permitted, and there are an infinite number of jobs at the first station. They showed that if service times are stochastically identical for all tasks, or for all workers, then any nonidling policy is optimal. For two stations, two workers, and exponential service times with rate μ_{ij} for worker i processing task j, where $\mu_{11}\mu_{22} \geq \mu_{12}\mu_{21}$, it is optimal to assign worker one to 1-tasks and worker two to 2-tasks when that assignment is nonidling; otherwise, the workers should collaborate on tasks in the nonempty buffer.

Van Oyen *et al.* [28] considered a system in which workers are identical, fully trained, and can collaborate on tasks. They showed that when job completions must be in FCFS order from the last station, the expedite policy (that all workers should work on the most downstream job, following it to completion) minimizes the completion time for all jobs and sample paths. Note that the expedite policy is the same as the LBFS policy. For closed queueing networks and exponential processing times, the expedite policy minimizes the mean cycle time. They also studied the pick-and-run policy when workers cannot collaborate. In the pick-and-run policy workers follow jobs to completion before starting a new job, so this policy is equivalent to LBFS when workers are identical and service times are exponential. In this case they showed that the pick-and-run policy minimizes the mean cycle time. They quantified the advantage, in mean cycle time, of having fully cross-trained workers over the best static allocation of workers.

Bartholdi and Eisenstein [9] and Bartholdi *et al.* [10] studied a bucket brigade arrangement of workers when all workers are fully trained and worker speeds are uniformly ordered (so, for example, if one worker is on average twice as fast at one task than another worker, then it is twice as fast at all tasks). In a bucket brigade the w workers are arranged from slowest to fastest and the ith (fastest) worker is always working on the job that has completed the ith most tasks, i.e. the ith furthest downstream job, among those present. When a downstream worker completes a job or is bumped by a further downstream worker, it moves back up the line and takes over the task of the next upstream worker. Bartholdi *et al.* also assumed that multiple workers cannot work on jobs at the same station at the same time, and that upstream workers that become blocked by downstream workers must idle. The Toyota Sewn Products Management System operates as a bucket brigade system, except that workers in the system may be restricted to zones and are not necessarily ordered from slowest to fastest. Bartholdi and Eisenstein [9]

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

showed that, for deterministic processing times, when workers are arranged from slowest to fastest, in the long run bucket brigades achieve the maximum possible service rate and workers settle into 'virtual zones' of operation. For other worker arrangements the system may not be stable even when the arrival rate is less than the total service rate. Bartholdi *et al.* [10] showed that for exponential processing times, as the number of tasks gets large the system's behavior approaches that of the deterministic system.

When there is zone cross-training the problem is much harder, and most of the literature studies heuristics using simulation models. Ostolaza *et al.* [23], McClain *et al.* [22], Askin and Chen [6], [7], and Gel *et al.* [14] considered finite-buffer systems with zone cross-training in closed dynamic assembly-line balancing models. Each worker has a fixed task that only that worker can do, and each sequential pair of workers can do a shared task, so $k_1 = 1$, $l_1 = 2$, $k_i = l_{i-1}$ for $i = 2, \ldots, w$, $l_i = k_i + 2$ for $i = 2, \ldots, w-1$, and $l_w = k_w + 1$ (in [14] and [6], $w = 2$ and $n = 3$). Once a worker passes a job down to the next worker, it cannot work on it again. The cited authors studied heuristic rules to determine whether a worker that has just finished its fixed task for a job should pass the job down to the next worker to do the shared task, or do the shared task itself. These rules are typically based on the number of jobs waiting at the downstream worker. Many of these authors also studied the effects of the amount of cross-training and the placement of buffers on throughput. The paper of Hopp *et al.* [17] is similar in spirit, but they studied both a zoned model of cross-training, where $k_1 = 1$, $l_1 = D$, and $k_i = k_{i-1} + 1$ and $l_i = l_{i-1} + 1$ for $i = 2, \ldots, w$ (called D-chaining), and a nonzoned model called cherry picking. They looked at many different heuristics for dynamically assigning workers to tasks. Koole and Righter [20] considered zone cross-training with no shared tasks, so $k_1 = 1$ and $k_i = l_{i-1} + 1$ for $i = 2, \ldots, w$. They showed that the LBFS policy, which in this case is equivalent to the pick-and-run policy, is often optimal.

In the serial model with general zone cross-training, optimal policies are often complicated and difficult to compute and implement. In this paper we attempt to identify conditions under which simple policies will be optimal. In Section 2 we consider fixed tasks only, but otherwise permit very general assumptions on processing times, the number of stages, arrival processes, worker capabilities, and objective functions. We show that in these general systems it is often optimal for workers to follow the LBFS policy for their fixed tasks (pick-and-run). To better understand the optimal policy for shared tasks, we restrict ourselves to deterministic or exponential processing times for the remainder of the paper. In Section 3 we assume that processing times are deterministic and give conditions for the optimality of the FBFS and LBFS policies, respectively. We consider exponential processing times in Section 4. We show that under very general conditions there will be a tendency for workers to work together (either collaboratively on the same job, if permissible, or noncollaboratively on different jobs at the same station) under the optimal policy. We also give conditions such that, when workers are fully trained and multiple workers are permitted at the same station (which is not generally true for standard bucket brigades), a bucket brigade-type scheduling rule with workers arranged from slowest to fastest maximizes the mean flowtime for the first k jobs, for any k. This rule is equivalent to the LBFS policy with priority given to faster workers.

Throughout, we use the terms 'larger', 'increasing', etc. in the nonstrict sense.

2. Fixed tasks and general processing times

Recall that a job consists of tasks that must be done sequentially, so we think of the jobs as moving through a tandem system of stations, or a set of buffers in tandem. Also, workers can do some sequential subset of the tasks, i.e. worker i can work on jobs in buffers $k_i, k_i+1, \ldots, l_i$ for

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

some k_i and l_i, where k_i and l_i are increasing in i. If the workers have different speeds then faster workers have priority over slower ones; otherwise, an arbitrary priority order is given. Under the FBFS policy worker i always works on the task in its first nonempty buffer that is unclaimed by higher priority workers, i.e. it works on task j where $j = \arg\min\{k\colon k_i \le k \le l_i,\ n_k > 0\}$ and n_k is the number of tasks in buffer k that have not been assigned to higher priority workers. The worker idles if $n_k = 0$ for all k, $k_i \le k \le l_i$. When workers can collaborate we need not worry about jobs being claimed by higher priority workers; they can work together on tasks in their last nonempty buffer. The LBFS policy is similarly defined.

We call a worker an exclusive worker if it is the only one that can do some of the tasks (though it may also be able to do some shared tasks). That is, worker i is an exclusive worker if $l_{i-1} < k_{i+1} - 1$, so it is the only one that can do tasks in buffers $l_{i-1}+1, \ldots, k_{i+1}-1$; we call these tasks its fixed tasks. We first consider the optimal policy for the fixed tasks of exclusive workers. For these tasks there is no overtaking, in the sense that if a worker works on one of these tasks for a given job, then no other job can overtake the given job. Under these conditions we can otherwise have very general assumptions on processing times and worker availabilities. Our results extend single-server scheduling results. In particular, the LBFS policy corresponds to the shortest expected remaining processing time policy. We give conditions for the optimality of the LBFS policy among fixed tasks. This means, perhaps after some initial clearing, that workers will follow jobs through their fixed tasks (pick-and-run).

Processing times may have an arbitrary distribution that might depend on both the task and the worker. Workers may fail (e.g. go on break) and be repaired (return), with random state-independent failure and repair times, and arrivals are state independent but otherwise arbitrary. The system may be either open or closed. We first assume that workers cannot collaborate, i.e. that two workers cannot work on the same job at the same time.

Let c_t^j be the number of task-j completions by time t, and let $\bar{C}_t^j = \sum_{i=j}^n c_t^i$ be the number of completions of task j or those that are downstream of j by time t. We call $\{\bar{C}_t\}_{t=0}^\infty = \{\bar{C}_t^1, \bar{C}_t^2, \ldots, \bar{C}_t^n\}_{t=0}^\infty$ the joint task completion process and $\{\bar{C}_t^1\}_{t=0}^\infty$ the total task completion process. The job completion process is therefore $\{c_t^n\}_{t=0}^\infty = \{\bar{C}_t^n\}_{t=0}^\infty$, and maximizing this implies minimizing the number of jobs in the system, minimizing the flowtimes or cycle times of jobs, and maximizing throughput. Note also that stochastically maximizing the joint task completion process implies that the total reward process is also stochastically maximized. We write $r(j)$, which is increasing, for the reward for completion of a j-task. Maximizing the total task completion process is equivalent to maximizing worker utilization or the proportion of time workers are busy. For stable systems, this implies the maximization of long-run throughput. Let $h(j)$ be the holding cost per task per unit time for j-tasks, and assume that h is finite, increasing (so value is added as jobs get closer to full completion), and concave. Let H_t be the total holding cost at time t, and $\{H_t\}_{t=0}^\infty$ be the holding cost process.

We say that the random variable X is larger than the random variable Y in the stochastic ordering sense, $X \ge_{\mathrm{st}} Y$, if $F(t) \le G(t)$ for all t, where $F(t)$ and $G(t)$ are the cumulative distribution functions of X and Y, respectively. We say that X is larger than Y in the likelihood ratio ordering sense, $X \ge_{\mathrm{lr}} Y$, if $f(t)/g(t)$ is increasing in t, where $f(t)$ and $g(t)$ are the densities or probability mass functions of X and Y, respectively. A random variable X is increasing in likelihood ratio (ILR) if $X_t \le_{\mathrm{lr}} X_s$ for $s \le t$, where $X_t = [X - t \mid X \ge t]$. Thus, a task processing time that is ILR is getting smaller in a strong sense as the task is being served, so there is an incentive not to preempt tasks with other tasks at the same station (i.e. to follow FCFS) when processing times are ILR.

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

Theorem 2.1. *Consider a noncollaborative, open or closed tandem system.*

(i) *Suppose that preemption and idling are permitted. Suppose that processing times for fixed tasks are ILR, or that tasks within a fixed-task buffer must be served according to FCFS. To maximize the job completion process, workers should never idle and exclusive workers should follow LBFS and FCFS within a buffer for their fixed tasks. For any policy that violates these properties, there exists an alternative policy that does not, such that* $\{c_t^n\}_{t=0}^{\infty}$ *is stochastically larger for the alternative policy.*

(ii) *Suppose that preemption and idling are not permitted. Suppose that, for each exclusive worker, the processing times of all fixed tasks have the same (not necessarily ILR) distribution. To maximize the joint task completion process and minimize the holding cost process, exclusive workers should follow LBFS for their fixed tasks. For any policy that violates this property, there exists an alternative policy that does not, such that* $\{\bar{C}_t\}_{t=0}^{\infty}$ *is stochastically larger and* $\{H_t\}_{t=0}^{\infty}$ *is stochastically smaller for the alternative policy.*

Proof. (i) First suppose that processing times are ILR. We must actually show a stronger result than the optimality of FCFS within a buffer: the task with the most-attained service should instead have priority within a buffer. This is because, under an arbitrary policy where preemption is permitted, at any given time there may be several jobs in the same buffer whose tasks there have been partially completed. Note that, under the most attained service time (MAST) policy, the first time there are no partially completed tasks, the policy corresponds to a nonpreemptive policy (e.g. FCFS) from then on.

Let us call a policy that never idles, follows MAST within fixed buffers, and follows LBFS for the fixed tasks for all exclusive workers, an M–L policy. We assume that time is discrete, where the discretization can be arbitrarily small. Our proof is by induction on a finite time horizon, T. Suppose that an M–L policy is optimal for $T-1$, and consider T. (The result for $T=1$ is easy using the arguments below.) Let π be an arbitrary policy that is not an M–L policy for time horizon T. We will construct an M–L policy $\hat{\pi}$ that is better than π for our objective. We may assume from the induction hypothesis that π is an M–L policy from time 1 on, since otherwise the policy that agrees with π at time 0 and then follows an M–L policy is better than π.

Nonidling: If π idles any worker at time 0, it is easy to construct a policy that does not and that is better than π, so let us suppose that π is nonidling.

MAST: Suppose that, at time 0 under π, an exclusive worker, l, serves a job J in one of its fixed buffers, k, when there is a job I in buffer k that has received more service in buffer k than has job J. Then π will not serve J again until I is served, say at time σ for the first time. Let $\hat{\pi}$ agree with π except in that it serves I at time 0 and J at time σ, and thereafter serves I or J with priority to I whenever π serves I or J. We also assume that $\hat{\pi}$ agrees with π for the other workers besides l, with one possible small exception which will be given below. Let X and Y be the remaining task processing times of jobs I and J at time 0, where, by assumption, $X \le_{\mathrm{lr}} Y$. We will couple X and Y under π with $\hat{X}$ and $\hat{Y}$ under $\hat{\pi}$, as follows.

First, generate $m = \min\{X, Y\}$ and $M = \max\{X, Y\}$ with the appropriate distribution. If $m = M = 1$ then let $X = \hat{X} = Y = \hat{Y} = 1$, in which case the states and task completions under π and $\hat{\pi}$ will be the same. If $1 < m \le M$ then let $X = \hat{X} = m$ and $Y = \hat{Y} = M$ with probability

$$p := \mathrm{P}(X = m,\ \ Y = M \mid \min(X, Y) = m,\ \ \max(X, Y) = M)$$

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

and let $X = \hat{X} = M$ and $Y = \hat{Y} = m$ with probability $1 - p$. The states and task completions under π and $\hat{\pi}$ will again be the same. Finally, suppose that $1 = m < M$. It is easy to show that $p \geq 1 - p$ for $X \leq_{\text{lr}} Y$. In this case let $X = \hat{X} = 1$ and $Y = \hat{Y} = M$ with probability $1 - p$, let $X = \hat{X} = M$ and $Y = \hat{Y} = 1$ with probability $1 - p$, and let $\hat{X} = Y = 1$ and $X = \hat{Y} = M$ with probability $2p - 1$. For the first two subcases the interchange has no effect and again the states and task completions under both policies will be the same. In the third subcase, let us interchange the labels of jobs I and J under $\hat{\pi}$. Then job J completes task k at time $\sigma + 1$ under π and at time 1 under $\hat{\pi}$, while all other task completions for worker l, and the states from time $\sigma + 1$ on, are the same under both policies. If task $k + 1$ is not a fixed task for l (i.e. it is shared with other workers), then under $\hat{\pi}$ it is assumed that the other workers cannot work on job J until time $\sigma + 1$. For our objective, $\hat{\pi}$ will be better than π. Note that $\hat{\pi}$ may idle a worker at time $\sigma + 1$; from the induction hypothesis we can then construct another policy that agrees with $\hat{\pi}$ until time $\sigma + 1$, but does not idle at that time, that is better under our objectives.

LBFS: If, at time 0 under π, an exclusive server, l, serves a job I in fixed buffer i when there is a job J in fixed buffer $j > i$, then a simple interchange will give us a better policy for our objectives. The same interchange argument also gives us our result for non-ILR processing times when service within a buffer is required to be FCFS.

(ii) This result also follows from a simple interchange argument.

Note that, in (i), because the optimal policy is nonpreemptive and nonidling, it will also be optimal when preemption and idling are not permitted and when processing times are ILR and not necessarily identical.

The optimality of LBFS for the fixed tasks of individual exclusive workers can be extended to collaborative teams of heterogeneous workers in the following discrete-time model. Suppose that all the workers within a team can do the same subset of tasks, and the teams have zone cross-training. A team is called an exclusive team if it has fixed tasks, i.e. tasks that no other teams are trained to do. (Note that now fixed tasks can be done by more than one worker, but that all the workers that can do a fixed task must be on the same team.) Teams can work collaboratively on the same job, but members of different teams cannot collaborate. Time is discrete (with arbitrarily small time intervals) and $\gamma_j(t)$ denotes the nominal hazard rate function of task j. That is, if a job has received t service units for task j without having completed task j, and receives one service unit for the current time slot, then it will complete with probability $\gamma_j(t)$ in that time slot. For a fixed team i, let w_i be the number of members of the team and let ν_{ikj} be the speed, or amount of service in a time slot, of the kth team member of team i when performing task j, where we assume that $\max_{i,j}\{\sum_{k=1}^{w_i} \nu_{ikj}\} \leq 1$. Then, if a subset S of team i is collaboratively working on task j of a job with t completed service units for that task, the probability that the task will complete in the current time slot is $\sum_{k \in S} \nu_{ikj} \gamma_j(t)$. If it does not complete, then it will have received a total of $t + \sum_{k \in S} \nu_{ikj}$ service units by the end of the slot. The argument for the following corollary is similar to that for Theorem 2.1. The corollary generalizes a result of [28], where the expedite policy (which for fixed tasks is equivalent to LBFS) was shown to be optimal for a single, fully cross-trained collaborative team of identical workers when jobs within a buffer must be served according to FCFS.

Corollary 2.1. *Consider a collaborative, open or closed tandem system.*

(i) *Suppose that preemption and idling are permitted. Suppose that processing times for fixed tasks are ILR, or that tasks within a fixed-task buffer must be served according to FCFS. To*

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

maximize the job completion process, workers should never idle and exclusive workers should follow LBFS and FCFS within a buffer for their fixed tasks. For any policy that violates these properties, there exists an alternative policy that does not, such that $\{c_t^n\}_{t=0}^{\infty}$ *is stochastically larger for the alternative policy.*

(ii) *Suppose that preemption and idling are not permitted. Let* S_i *be the set of fixed tasks of exclusive collaborative team* i*, and suppose that for each team* i *the processing times for fixed tasks are identically distributed in the sense that* $\gamma_j(t) = \gamma_l(t)$ *for all* t *and all* $j, l \in S_i$*, and that* $v_{ikj} = v_{ikl}$ *for all* $j, l \in S_i$ *and for each member* k *of team* i*. To maximize the joint task completion process and minimize the holding cost process, exclusive workers should follow LBFS for their fixed tasks. For any policy that violates this property, there exists an alternative policy that does not, such that* $\{\tilde{C}_t\}_{t=0}^{\infty}$ *is stochastically larger and* $\{H_t\}_{t=0}^{\infty}$ *is stochastically smaller for the alternative policy.*

In this section we have determined the optimal policy for fixed tasks for very general processing times. The optimality of LBFS for fixed tasks is analogous to the optimality of the smallest expected remaining processing time policy in single-server queueing systems. To study good policies for shared tasks (where we have parallel servers) we specialize to deterministic and exponential processing times in the next two sections, respectively.

3. Shared tasks and deterministic processing times

Suppose that time is discrete and that all processing times are deterministic, identical, and of length 1. We can assume this without too much loss of generality, because we can redefine an integer-valued processing time of length l as a series of l unit-length subtasks, though in this case we must allow preemption of tasks. We assume that workers have identical speeds and are always available, and that collaboration is not permitted. We also assume that idling is permitted unless we specifically disallow it, and that preemption can only occur at integer time points. In this section we first consider open systems where the number of arrivals in each slot or period (at each time unit) forms an arbitrary random process that is independent of the state of the system. For this slotted model, we assume that we first observe the state at the beginning of the slot and then make our decision. At the end of the slot arrivals and task completions occur and the holding cost for the slot is incurred.

In the last section we showed that, for exclusive workers, i.e. those that have tasks that only they can do (their fixed tasks), it is optimal to follow LBFS for the fixed tasks. Recall that worker i can work on jobs in buffers $k_i, k_i + 1, \ldots, l_i$ for some k_i and l_i, where k_i and l_i are increasing in i, and that worker i is an exclusive worker if $l_{i-1} < k_{i+1} - 1$, meaning that it is the only one that can do tasks in buffers $l_{i-1} + 1, \ldots, k_{i+1} - 1$ (its fixed tasks). For these tasks there is no overtaking. We now wish to study the optimal policy for shared tasks, i.e. those that more than one worker can do and for which there is the possibility of overtaking. Suppose that worker i is an exclusive worker ($l_{i-1} < k_{i+1} - 1$) and that it has at least one shared task before its fixed tasks ($k_i \le l_{i-1}$, so the zones for workers $i - 1$ and i overlap). Then the last shared task before its fixed tasks is task l_{i-1}. In the case of deterministic processing times, we can extend the optimality of LBFS for exclusive workers to the last shared task before their fixed tasks, i.e. LBFS is optimal for worker i for tasks $l_{i-1}, l_{i-1} + 1, \ldots, k_{i+1} - 1$.

Lemma 3.1. *To maximize the joint task completion process, workers should never idle, exclusive workers should follow the LBFS policy among their fixed tasks, and the fixed tasks for each exclusive worker should have priority over the last shared task before its fixed tasks. For any*

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

policy that violates these properties, there exists an alternative policy that does not, such that $\{\bar{C}_t\}_{t=0}^{\infty}$ *is stochastically larger for the alternative policy.*

Note that if an exclusive worker has only one shared task, and it is before its fixed tasks, then this policy is consistent with the fixed before shared policy of [15].

3.1. Two workers with shared and fixed tasks

To investigate good policies for shared tasks, we simplify our model to describe two always-available workers: worker one can do the first two tasks and worker two can do the last $n-1$ tasks (or, equivalently, can do a third task of length $n-2$), so task 2 is a shared task. Arrivals may be arbitrary and random, but are independent of the state of the system. Thus, the workstation consisting of these two workers and their n tasks may be part of a larger stochastic network. Let n_i be the number of tasks in buffer i. For this system, when we say that a policy is nonidling, we assume that when $n_2 = 1$ and $n_i = 0$ for $i > 2$ the second worker works on the 2-task (and the first worker works on 1-tasks, if any).

Theorem 3.1. *When there are two always-available workers, worker one can do the first two tasks and worker two can do the last* $n-1$ *tasks* ($n > 2$)*, the nonidling LBFS policy stochastically maximizes the joint task departure process. If idling is not permitted, the LBFS policy stochastically minimizes the total holding cost process.*

Proof. From Lemma 3.1, we need only show the optimality of LBFS for worker one. Our proof is by induction on a finite time horizon, T. Suppose that the result holds for $T-1$, and consider T. (The result for $T = 1$ is trivial.) It is easy to show that if $n_2 = 1$ and $n_i = 0$ for $i > 2$ (or, equivalently, $\sum_{i=3}^{n} n_i = 0$), then the second worker should serve the 2-task.

Let π be an arbitrary policy that disagrees with LBFS for time horizon T. We will construct a policy $\hat{\pi}$ that agrees with LBFS and that is better than π for our objectives. We may assume from the induction hypothesis that π agrees with the LBFS policy from time 1 on, since otherwise we can construct such a policy that is better than π. We may also assume that π agrees with the LBFS policy from time 0 on for worker two. Therefore, suppose that under π worker one serves a 1-task at time 0 when a 2-task is available, so we can rule out the case $n_2 = 0$, as well as the case with $n_2 = 1$ and $\sum_{i=3}^{n} n_i = 0$. Thus, $n_2 > 1$ or $\sum_{i=3}^{n} n_i > 0$ and $n_2 = 1$. Then, at time 1 under π, worker one will serve a 2-task (possibly generated by the 1-task served at time 0) and worker two will also have a task to serve. Let $\hat{\pi}$ have worker one serve the 2-task at time 0 and the 1-task at time 1. Let us couple the arrival processes so that they are the same under both policies.

First, suppose that $n_2 > 1$, $\sum_{i=3}^{n-1} n_i > 0$, or $n_n > 1$. Then we can let $\hat{\pi}$ agree with π on the decisions made at times 0 and 1 for worker two, and for all decisions made after time 1. The states will be the same at time 2 under both policies, and, with obvious extensions of our earlier notation, $\bar{C}_t = \hat{C}_t$ and $H_t = \hat{H}_t$ for all $t \neq 1$, $\hat{c}_1^1 = c_1^1 - 1$, $\hat{c}_1^2 = c_1^2 + 1$, $\hat{c}_1^j = c_1^j$ for $j \neq 1, 2$, and $\hat{H}_t = H_t + h(1) - 2h(2) + h(3) < H_t$ from concavity.

Now suppose that $n_2 = n_n = \sum_{i=3}^{n} n_i = 1$. Then, under policy π, worker two will serve the n-task at time 0, the 2-task at time 1, and the corresponding 3-task at time 2. Let $\hat{\pi}$ have worker two serve the n-task at time 0, the 3-task (generated by worker one serving the 2-task at time 0) at time 1, and the 2-task (generated by the 1-task served by worker one at time 1) at time 3. Then the states will be the same under both policies from time 4 on, and before that time $\hat{\pi}$ will be better than π, arguing as before.

Note that $\hat{\pi}$ agrees with LBFS at time 0, but may not agree with it thereafter. By the induction hypothesis, the LBFS policy will be better than $\hat{\pi}$.

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

Note that, under this policy, both workers will pick-and-run, with worker two following a job through all of its tasks, starting with task 2 or 3, and worker one starting a job with task 1 and then doing task 2 for the same job, unless worker two would be idle. Thus, this policy satisfies the additional constraints that worker one cannot resume a job that it hands off to worker two, and worker two cannot preempt a job. Therefore, the policy is still optimal when these additional constraints are imposed, as in some dynamic assembly-line balancing systems.

If worker one can do the first $l > 2$ tasks and worker two can do the last $n - l + 1$ tasks (so both workers can do task l), we know from Lemma 3.1 that LBFS is optimal for worker two and that, for worker one, LBFS is optimal for the first $l - 1$ tasks. However, there will be situations in which worker one should give priority to earlier tasks over l-tasks. Moreover, the LBFS policy is no longer optimal if worker one can do all tasks or if both workers can do all tasks (even when there are only two tasks). Consider the system with two fully trained workers, $n = 3, n_1 = 0, n_2 = 1, n_3 = 2$, and no arrivals. Under the LBFS policy, both workers will do 3-tasks at time 0, and then one worker will idle at times 1 and 2 while the other does a 2-task and then a 3-task. If at time 0 one worker does the 2-task while the other does a 3-task, then both can work on 3-tasks at time 1 and the system will be empty at time 2.

3.2. Fully trained workers

When all workers can do all tasks, FBFS for all workers is optimal in the more restricted sense of stochastically maximizing the total number of task completions (and keeping the workers busy). Therefore, it also stochastically minimizes the total remaining work in the system. Workers may now be randomly unavailable during a time slot, and we assume that a worker is either available or unavailable for the entire slot and that its availability is known at the beginning of the slot.

Theorem 3.2. *For systems with multiple fully trained workers and randomly unavailable workers, the nonidling FBFS policy stochastically maximizes the total task completion process,* $\{\bar{C}_t^1\}_{t=0}^{\infty}$.

Proof. We again use induction on the time horizon, T. Suppose that the result holds for $T - 1$, and consider T. (The result for $T = 1$ is trivial.) It is easy to show that the optimal policy is nonidling.

Let π be an arbitrary policy that disagrees with FBFS for time horizon T, though as before we may assume that it agrees with FBFS from time 1 on. Suppose that at time 0 policy π serves job J in buffer j and not job I in buffer i, where $i < j$. From time 1 on, π (and FBFS) will not serve J at a given time without also serving I at that time until I is served alone at least once. Thus, under π, at some time t job I will be served alone and between times 1 and t job J will not be served alone. Let $\hat{\pi}$ agree with π except in that at time 0 it serves I and at time t it serves J. Then, at time $t + 1$, the states will be the same under both policies, and the task completion and work processes will also be the same. It may be that at time t a worker is idle under both π and $\hat{\pi}$ (if J has completed by time t under π), in which case a policy $\tilde{\pi}$ that serves I at time t and otherwise agrees with $\hat{\pi}$ will be better than $\hat{\pi}$. By the induction hypothesis, FBFS will be better than $\tilde{\pi}$.

If the arrival rate λ is less than the total service rate (w/n), then FBFS results in a stable system and achieves the maximal overall throughput (rate of completions of n-tasks) of λ. However, it will not achieve maximal overall throughput (of w/n) for unstable systems, because servers will be kept busy at earlier buffers and jobs will tend not to be completed. In the extreme case of an infinite supply of jobs at station 1, the overall throughput is 0 under FBFS (though the

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

servers will always be busy). Indeed, it is easy to show that, for an infinite supply of jobs (or a closed system), the LBFS (pick-and-run) policy for fully trained workers stochastically maximizes the joint task completion process and stochastically minimizes the job holding cost process $\{\mathcal{H}_k\}_{k=1}^{\infty}$, where $\mathcal{H}_k$ is the total holding cost for the first k jobs to depart.

With a general arrival process and fully trained workers, we can improve upon the nonidling FBFS policy under the more general objectives of maximizing the joint task completion process and minimizing the holding cost, without reducing the total task completion process. Let FBFS/LBFS be the nonidling policy whereby $w-1$ workers follow the FBFS policy (call them the F-workers) and have priority over one of the fully trained workers, which follows the LBFS policy (call it the L-worker). The L-worker at the end of the line is an anchor worker who is fully cross-trained, as in a bucket brigade.

Theorem 3.3. *When workers are fully trained and always available, the FBFS/LBFS policy has a stochastically larger joint task completion process and a stochastically smaller holding cost process than does the FBFS policy.*

Proof. We again use induction on the time horizon, T (and the $T = 1$ case is trivial).

Let the L-worker be the one with lowest priority under the FBFS policy, so all other workers are either upstream of the L-worker or at the same station. Denote by I, in buffer i, the task that the L-worker serves at time 0 under FBFS, and denote by J, in buffer j, the job that is in the latest nonempty buffer. Suppose that $i \neq j$ (otherwise, the policies coincide at time 0). Let π agree with FBFS at time 0 and, again without loss of generality from the induction hypothesis, let it agree with FBFS/LBFS from time 1 on. Thus, under π the L-worker serves I at time 0 and J at time 1. Also, I will not be served by the F-workers at time 1, because each of them will serve a job in an earlier buffer or will continue serving the same job it served at time 0. Thus, we can do a simple interchange, letting $\hat{\pi}$ have the L-worker serve J at time 0, serve I at time 1, and otherwise agree with π. Thus, under $\hat{\pi}$, $\{\bar{C}_t\}_{t=0}^{\infty}$ will be larger and $\{H_t\}_{t=0}^{\infty}$ smaller, because of the concavity of holding costs. Again, by induction, the FBFS/LBFS policy at all times will be even better than $\hat{\pi}$. (Note that $\{\bar{C}_t^1\}_{t=0}^{\infty}$ is the same under both π and $\hat{\pi}$.)

Now let us consider closed systems. It is not hard to see that Lemma 3.1 and Theorem 3.1 still hold. Indeed, for closed systems Theorem 3.1 can be extended to show that LBFS is optimal when worker one can do the first l tasks and worker two can do the last k tasks, with $l + k \geq n$. As indicated earlier, for fully trained workers LBFS stochastically minimizes the joint task completion process and the job holding cost process.

4. Shared tasks and exponential processing times

In this section we consider both open and closed systems, and for open systems we assume that the arrival process is a general Markov arrival process. That is, there is an environmental continuous-time Markov chain with transition rates α_{xy} from state x to state y, such that arrivals occur at x-to-y transitions with probability β_{xy}. Such processes are dense in the class of arbitrary arrival processes [8]. We assume that $\sum_y \alpha_{xy}$ is bounded by some $\bar{\alpha}$ for all x. Workers may fail (or become unavailable): worker k fails after working for an exponential time with rate ν_k and its repair (or return) time is exponential with rate υ_k. The processing time of task i by worker k, assuming that worker k is trained to do task i, is exponentially distributed with rate $\mu_k\eta_i$. Idling and preemption of tasks are permitted, unless otherwise stated. Let us add an additional task, $\mathcal{I}$, with rate $\eta_{\mathcal{I}} = 0$, corresponding to idling, and assume there are an infinite number of dummy jobs at station $\mathcal{I}$. We will use uniformization, with a uniformization rate

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

$\rho = \bar{\alpha} + \sum_{k=1}^{c} \mu_k \eta + \sum_{k=1}^{c} (\nu_k + \upsilon_k)$, where $\eta = \max_i \eta_i$, and we scale time so that $\rho = 1$. Thus, decision epochs occur according to a Poisson process with rate 1; we will call these decision epochs time points, where the initial time point is 0. Let s be the initial state, which consists of the conditions of all the workers (working or failed), the state of the environment, and the number of jobs at all the stations.

4.1. General systems

Here we may have arbitrary, not necessarily tandem, deterministic routeing of jobs through tasks, and arbitrary, not necessarily sequential or zoned, training of workers. The holding cost rate for each job at station j, $h(j)$, now is arbitrary, i.e. we relax the condition that it is increasing and concave, though it is still assumed to be finite. For each worker i, we also permit the worker's speed, μ_i, and competencies, S_i, to change according to a Markov process X_t. Here S_i is the set of tasks that worker i can do, including idling (task $\mathcal{I}$), so $S_i = \mathcal{I}$ corresponds to a failed, or unavailable, worker. Therefore, the state s now also includes the state of X_t, and we write $\mu_i(s)$ and $S_i(s)$, although μ_i and S_i depend only on the X_t part of the state. We include both the maximal rate of change in X_t and the maximal values of $\mu_i(s)$ in the uniformization rate ρ. For these general exponential systems, optimal policies will tend to have workers working at the same station, as long as the objective function is such that optimality can be achieved. Note that the Markov process X_t allows us to model training, for example, to change the set of tasks a worker can perform or the speed at which it can perform them.

Let us first consider the case when collaboration is not permitted.

For an arbitrary state s, let $V_t^{\pi}(s)$ be the total holding cost under some policy π for the next t decision epochs starting in state s. Note that $V_t^{\pi}(s)$ is a random variable. We first suppose that there exists a stochastically optimal policy π^*, i.e. one such that $V_t^{\pi^*}(s) \leq_{\mathrm{st}} V_t^{\pi}(s)$ for all t and s; let $V_t(s) \equiv V_t^{\pi^*}(s)$ be its total holding cost. Later we discuss the case of minimizing the expected total holding cost when stochastic minimization is not possible.

Let s_k be a state that is the same as s except in that a job in buffer k has finished its kth task (so $s_{\mathcal{I}} = s$), and let J_k be a Bernoulli random variable with $\mathrm{P}(J_k = 1) = \eta_k/\eta$. Let $g(s)$ be the total holding cost from time 0 to time 1 when the state is s. Let $w_i(s)$ be the number of workers that can do task i and let $n_i(s)$ be the number of jobs in buffer i in state s. The results below follow from [1]. They tell us that the optimal policy follows a state-dependent index rule and that there will be a tendency for workers to work together.

Theorem 4.1. *For state s, suppose that tasks i and j are such that*

$$J_i[V_{t-1}(s_i) - V_{t-1}(s)] \leq_{\mathrm{st}} J_j[V_{t-1}(s_j) - V_{t-1}(s)].$$

If a worker, worker one say, can do both tasks i and j in state s, then it will not be optimal to assign worker one to a job in buffer j while leaving a job in buffer i unassigned. If two workers, workers one and two say, can do both tasks i and j in state s, and if $\mu_1(s) \geq \mu_2(s)$, then it will not be optimal to assign worker one to a job in buffer j and worker two to a job in buffer i.

If a stochastically optimal policy does not exist, the result still holds, replacing random variables with their expected values.

Corollary 4.1. *Suppose that our objective is to minimize the expected holding cost. For state s, suppose that tasks i and j are such that*

$$\eta_i[\mathrm{E}\, V_{t-1}(s_i) - \mathrm{E}\, V_{t-1}(s)] \leq \eta_j[\mathrm{E}\, V_{t-1}(s_j) - \mathrm{E}\, V_{t-1}(s)].$$

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

If a worker, worker one say, can do both tasks i and j in state s, then it will not be optimal to assign worker one to a job in buffer j while leaving a job in buffer i unassigned. If both workers one and two can do both tasks i and j in state s, and if $\mu_1(s) \geq \mu_2(s)$, then it will not be optimal to assign worker one to a job in buffer j and worker two to a job in buffer i.

In general it will be hard to compute $\mathrm{E}\, V_t(s)$ or to order $J_i[V_{t-1}(s_i) - V_{t-1}(s)]$, but the following corollaries, especially Corollaries 4.4 and 4.6, help to reduce the number of states that we need to consider. In these corollaries we assume that a stochastically optimal policy exists. The corresponding results for minimizing expected holding costs also hold, with $\mathrm{E}\, V$ replacing V and η replacing J.

Corollary 4.2. *At time t in state s, order (relabel) the nonempty buffers (including the idling buffer $\mathfrak{I}$) in increasing stochastic order of $J_k[V_{t-1}(s_k) - V_{t-1}(s)]$. If $w_i(s) \leq n_i(s)$ for all i, then assigning each worker to the lowest-indexed task it is capable of doing stochastically minimizes the total holding cost.*

Corollary 4.3. *At time t in state s, order the jobs according to the buffer they are in (including w dummy jobs that are in the idling buffer $\mathfrak{I}$) in increasing stochastic order of $J_k[V_{t-1}(s_k) - V_{t-1}(s)$, and order the nonfailed workers in decreasing order of μ_i. If the workers are fully trained, then assigning the ith available worker to the ith job, for all available workers, stochastically minimizes the total holding cost.*

Let $\kappa(s)$ be the number of nonempty buffers in state s and let $n_{(i)}(s)$, $i = 1, \ldots, n$, be a reordering of the nonempty buffers such that

$$n_{(1)}(s) \leq n_{(i)}(s) \leq \cdots \leq n_{(\kappa)}(s).$$

In state s, let $\bar{n}(s)$ be the total number of jobs and let $w(s)$ be the number of nonfailed workers. The following corollary is immediate from Corollary 4.3. It allows us to significantly reduce the number of candidates for optimal assignments, for any state. In the corollary, when we say that available workers have adjacent speeds, we mean that when they are ordered according to their speeds they have consecutive indices.

Corollary 4.4. *For fully trained workers under the optimal policy, workers will be assigned to at most $b(s)$ buffers in state s, where*

$$b(s) = \min\left\{ j : \sum_{i=1}^{j} n_{(i)}(s) \geq \min\{\bar{n}(s), w(s)\} \right\}.$$

Thus, if $\min_i\{n_{(i)}(s)\} \geq w(s)$ then the workers will all be assigned to the same station. Also, under the optimal policy, there will be at most one buffer i such that the number of workers assigned to it is strictly between 0 and $\min(w(s), n_i)$. Finally, when multiple workers are assigned to the same station, they will have adjacent speeds.

For example, two fully trained workers will be assigned to the same station in all states such that $n_i \neq 1$ for all i. That is, for such states, when both workers are available the number of possible optimal actions is reduced from κ^2 to κ. For three available, fully trained workers, if there is at most one buffer with a single job, then servers will be assigned to at most two different stations, and the slowest and fastest will be assigned to the same station only if all three are assigned to that station.

When collaboration is permitted we have the results below, which also follow from [1].

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

Corollary 4.5. *Suppose that workers can collaborate but are not necessarily fully trained. Order (relabel) the nonempty buffers (including the idling buffer $\mathcal{I}$) in increasing order of $J_k[V_{t-1}(s_k) - V_{t-1}(s)$. The optimal assignment is to assign each worker to the lowest-indexed task it is capable of doing.*

Corollary 4.6. *When a subset of workers can do the same subset of tasks and can collaborate, they always work as a team.*

A consequence of Corollary 4.6 is that, for a subset of workers that can do the same subset of tasks and can collaborate, we may assume, without loss of generality, that they are a single worker.

As noted earlier, if there is no stochastically optimal policy then all the results in this section still hold, except with the random variables replaced by their means (e.g. with E V instead of V, η_k instead of J_k, etc.), and our objective is to minimize the expected holding cost. The results also easily extend to the infinite-horizon problem when the expected average cost criterion or the expected discounted cost criterion is considered, assuming that optimal solutions exist.

Andradóttir *et al.* [5] have shown that a generalized round robin policy for the type of model we consider here arbitrarily closely approximates the maximal achievable capacity.

4.2. Tandem systems with partial cross-training

We now return to our original, tandem, model. We make the same assumptions regarding server failures and the Markov arrival process (or a closed system) as in the previous section, but here assume that processing times depend only on the workers, not on the tasks (thus, $\eta_i \equiv \eta$).

4.2.1. *Systems with exclusive workers.* It is easy to strengthen Theorem 2.1 to show that an exclusive worker should follow LBFS when choosing among its fixed tasks and its last shared task before its fixed task (as in Lemma 3.1 for deterministic processing times). If an exclusive worker has only one shared task, and it is before its fixed task, then the policy for that worker is fully specified and is consistent with the fixed before shared policy of [15]. Note that for collaborative systems, from Corollary 4.6, any subset of workers that can do the same subset of tasks can be thought of as a single worker, and the team can therefore be thought of as an exclusive worker if it serves tasks that can be served by no other workers.

Lemma 4.1. *For open or closed systems, with or without collaboration, to maximize the joint task completion process, workers should never idle, exclusive workers should follow the LBFS policy among their fixed tasks, and the fixed tasks for each exclusive worker should have priority over the last shared task before its fixed tasks. For any policy that violates these properties, there exists an alternative policy that does not, and $\{\bar{C}_t\}_{t=0}^{\infty}$ is stochastically larger for the alternative policy.*

Proof. We again use induction on the time horizon, in this case the number of remaining 'uniformized' decision epochs, T. The result is easy for $T = 1$, so suppose that it holds for $T - 1$ remaining decision epochs, and consider T. We also again show that any policy π that does not follow LBFS for fixed tasks at time 0, but follows LBFS for those tasks thereafter, can be stochastically improved. Suppose that, at time 0, policy π has worker l serve job I in fixed buffer i, when job J is in l's fixed buffer $j > i$. Let $\hat{\pi}$ agree with π at time 0 except in that worker l serves job J. Let us couple events under the two policies so that if a worker completes a task under π, then it also completes the task under $\hat{\pi}$; the arrivals and server failure processes are similarly coupled. If the next event in the uniformized process is something other than l completing its task, then, upon letting $\hat{\pi}$ agree with π from then on, both policies have

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

the same states and joint task completion processes. If the next event is a task completion for worker l, then under $\hat{\pi}$ let worker l serve job I whenever it serves J under π, until π completes the j-task of job J. Note that π will not serve I again until the j-task of job J completes. Then task j of job J under π will complete at the same time as task i of job I under $\hat{\pi}$, when the states will be the same, and all other task completions will be the same under both policies. Hence, $\hat{\pi}$ will be better than π in terms of its joint task completion process.

We can fully characterize the optimal policy for the following two-worker partial-cross-training models. The systems may be either open or closed, and collaboration may or may not be permitted. From Lemma 4.1 we have the following corollary, and the subsequent theorem is easy to show.

Corollary 4.7. *If there is one flexible (fully trained) worker and the other worker can only do task 1, then the nonidling LBFS policy stochastically maximizes the joint task departure process. If idling is not permitted, then the LBFS policy stochastically minimizes the total holding cost process for open systems.*

Theorem 4.2. *If there are one flexible (fully trained) worker and two tasks, and the other worker can only do task 2, then the nonidling FBFS policy stochastically maximizes the total task completion process,* $\{\bar{C}_t^1\}_{t=0}^{\infty}$.

4.2.2. *Collaborative workers and nested zones.* We now suppose that workers can collaborate and that zones are nested in the sense that team i can do tasks $\{1, \ldots, l_i\}$ (so $k_i = 1$ for all workers i and $l_w = n$, where w is the number of teams of workers with the same skill set). These types of zones may occur, for example, when all newly hired employees first learn task 1, then task 2, etc. Nested zones are an example of a hierarchical cross-training structure. See [15] for real-world examples of such cross-training. We also suppose that, when two different teams work on the same task, they work collaboratively. We know from Corollary 4.5 that, for a holding cost objective, all teams should work on the lowest-indexed task they can and workers on the same team will always collaborate. We will show that, for the nested zone model of this section, to stochastically maximize $\{\bar{C}_t\}_{t=0}^{\infty}$, 'lowest indexed' always corresponds to 'latest buffer' and LBFS is again optimal.

Theorem 4.3. *For open or closed tandem systems with collaborative workers and nested zones, the (nonidling) LBFS policy stochastically maximizes the joint departure process,* $\{\bar{C}_t\}_{t=0}^{\infty}$.

Proof. We again use induction on T, and show that any policy π that does not follow LBFS at time 0, but follows LBFS thereafter, can be stochastically improved. Suppose that, at $t = 0$, policy π assigns a subset of workers Γ to a job I in buffer i although there is a job J in buffer $j > i$ that would be assigned to the workers in Γ under the LBFS policy. Let $\hat{\pi}$ agree with π at time 0 except in that workers in Γ serve J at time 0. If the task the workers in Γ are working on at time 0 does not complete at time 1, then the states will be the same under both policies at time 1, and, upon letting $\hat{\pi}$ agree with π from time 1 on, they will have the same joint task completion processes. If the task does complete then the states will be the same at time 1, except that under π job I will be in buffer $i + 1$ instead of in i and job J will be in buffer j instead of in $j + 1$. Let $\hat{\pi}$ agree with π from time 1 until task j of job J completes under π, except in that the workers that serve J under π serve I under $\hat{\pi}$ (which is possible because of the structure of the nested zones). Then the states under the two policies will be the same, except for the relative position of jobs I and J, until task j of job J completes under π. Note that, before job J completes task j, job I cannot overtake job J under π (i.e. under LBFS). It

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

is possible for I to enter buffer j before J leaves it, if $l_i = j - 1$ for some i, but at that point we may, without loss of generality, assume that only job J is served (collaboratively). Therefore, when task j of job J completes under π, the states under each policy will be the same. Before that time, $\hat{\pi}$ has a larger joint departure process than does π, and afterwards, upon letting $\hat{\pi}$ agree with π, the joint departure processes will be the same.

When all workers are fully trained, the LBFS policy corresponds to the expedite policy of [28].

A consequence of our result is a result by Mandelbaum and Reiman [21]. They considered a restricted form of collaboration, or resource pooling, in Jackson queueing networks. They compared steady-state sojourn times for a system with dedicated (exclusive) servers for each node in a network with those in one in which a single server with combined service rate can serve at all the queues. However, in their model, when service is pooled there can be no preemption and jobs are processed in FCFS order, so the pooled system operates as an M/PH/1 queue. Under these constraints the pooled model may have worse performance than the exclusive-server model. In the special case of tandem systems, Mandelbaum and Reiman showed that pooling is always better. This result follows from ours because, assuming that optimal policies are always followed, a system that permits collaboration and preemption and in which all servers can do all tasks (system 1, say) will perform better than a system that requires collaboration, does not permit preemption, and in which all servers can do all tasks (the M–R pooled system). System 1 will also perform better than a system that does not permit collaboration and in which each server can only do one task (the M–R dedicated-server system). From Theorem 4.3, the optimal policy for system 1 is the LBFS policy, which is equivalent to the nonpreemptive single-server policy when all servers can do all tasks. Hence, for tandem queues, the performance of the M–R pooled system is as good as that of system 1 and, hence, is better than that of the M–R dedicated-server system.

4.3. Tandem systems with full cross-training

Let us now suppose that the c workers are fully cross-trained, do not collaborate, and do not fail. The problem is much more difficult when workers cannot collaborate, and we must make stronger assumptions and use a weaker objective function to obtain our results. We assume that there is an infinite supply of jobs at the first station, or that the system is closed with m jobs. (It is obvious, and follows from our results, that for the closed system we can assume there to be $\min\{m, w\}$ jobs and workers, where, for $w > m$, the slowest workers are not used.) Preemption and idling are permitted. It is intuitively clear, and easy to show, that with preemption it will never be optimal to idle. Our objective is to minimize the flowtime, or cumulative departure time, F_k, for the first k jobs to depart, for any k. That is, we minimize $F_k = \sum_{i=1}^{k} D_i$, where D_i is the departure time from the system of the ith job to depart. (In the closed system, we say that a job departs when it leaves buffer n and is fed back to buffer 1.) We show that the LBFS policy is optimal, meaning that the fastest worker is assigned to the task in the latest buffer, the next fastest to the next latest, etc. Faster workers have priority over slower workers if there are fewer jobs than there are workers. Our result provides some theoretical support for the value of a bucket brigade-type worker assignment system, as described in Corollary 4.8, below.

Theorem 4.4. *Assume that there are an infinite number of jobs at the first station or that the system is closed, and that workers are fully trained and cannot collaborate. The nonidling LBFS policy minimizes the mean flowtime process,* $\{\mathrm{E}\, F_k\}_{k=0}^{\infty}$, *when* $w = 2$, $n = 2$, *or* $\mu_i \equiv \mu$ *for any* n *and* w.

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

Proof. We first assume that $w = 2$, so potential service completions occur according to a Poisson process with rate $\mu_1 + \mu_2$, where we assume, without loss of generality, that $\mu_1 \geq \mu_2$ and the uniformization rate is $\mu_1 + \mu_2 = 1$. We again use induction on the time horizon, or remaining number of decision points, T. At the time horizon we suppose that all jobs immediately depart. To show that no worker should idle is easy. Suppose that, at time 0, the optimal policy π has worker one, say, serve job I in buffer i, and that job J in buffer j is not served, where $i < j$ and j is the largest such buffer. Then, again without loss of generality, under π, J will be served at time 1. Let $\hat{\pi}$ have worker one serve J at time 0; as before, we need only construct such a $\hat{\pi}$ with a smaller flowtime process than π. If worker one does not complete service at time 1, then the states will be the same under both policies (case (a), below). If worker one completes service and J departs ($j = n$), we have case (c), and if worker one completes service and J does not depart ($j < n$), we have case (b). Refer to Figures 1, 2, and 3, which show, for policies π and $\hat{\pi}$ and for each of the cases given below, the positions of jobs I and J and others that are treated differently under the two policies.

(a) The states are the same and, upon letting $\hat{\pi}$ agree with π, all future job departures will be the same.

(b) The states are the same except for jobs I and J: the downstream job J will have completed one more task under $\hat{\pi}$ than under π, and the upstream job I will have completed one more task under π than under $\hat{\pi}$. Let $\hat{\pi}$ agree with π from time 1 until either (i) job I 'passes' job J under π, i.e. job I is in the buffer after job J, (ii) job J completes its final task under $\hat{\pi}$, or (iii) we reach the time horizon. In case (i) under $\hat{\pi}$ job I will be in the same buffer that job J is under π, and vice versa, so the states will be the same because jobs are indistinguishable except for the tasks they have completed. That is, case (i) is the same as case (a). If we reach the time horizon (case (iii)) then all flowtimes are the same under the two policies. Case (ii) is the same as case (c), below.

(c) The state under $\hat{\pi}$ is the same as the state under π, except π has an extra job, J, in its last buffer, and job I has completed one more task under π than under $\hat{\pi}$. At this point, under π worker one will serve job J and worker two will serve job K, where K is the job that has completed the most tasks after J under π; this is also the job that has completed the most tasks under $\hat{\pi}$ (let $K = I$ if there is a tie). Let $\hat{\pi}$ agree with LBFS at this time, so that under $\hat{\pi}$ worker one serves job K and worker two serves job L, where L is the job that has completed the second-greatest number of tasks under $\hat{\pi}$. Let us couple the processing times such that if K completes under π then it also completes under $\hat{\pi}$, and if J completes under π then either K completes under $\hat{\pi}$, with probability $\mu_1 - \mu_2$, or L completes under $\hat{\pi}$, with probability μ_2. In the first case, when K completes under both policies, if $K = I$ and it is in the last buffer under π, then the states are the same under each policy (case (a)), with job J in the last buffer under π and job I in the last buffer under $\hat{\pi}$. Note that the first of the two jobs I and J to depart under the two policies does so earlier under $\hat{\pi}$; all other departures occur at the same time. If $K \neq I$ and K completes under both policies then the relative states under π and $\hat{\pi}$ are still in case (c), and we continue the same coupling until J completes under π or the time horizon is reached. When J completes under π and I completes under $\hat{\pi}$ (it is either job K or job L), the states will be the same under each policy (case (a)), the flowtimes for all jobs except J are the same, and the flowtime for job J is shorter under $\hat{\pi}$. If J completes under π and the job that completes under $\hat{\pi}$ is not job I, but is a job in a later buffer than is I (either K or L; see diagram (c-i) of Figure 2 for the job positions just before completion), then the relative states under π and $\hat{\pi}$ are the same as in case (b) or case (c) with job K or job L replacing job J, and

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

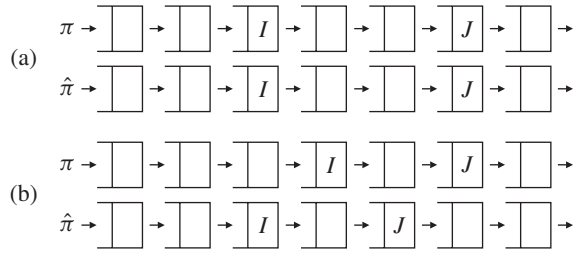

FIGURE 1: Job positions in cases (a) and (b).

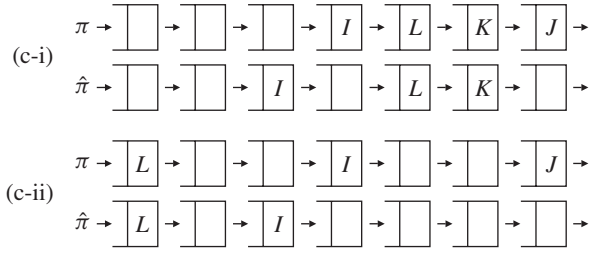

FIGURE 2: Job positions in case (c).

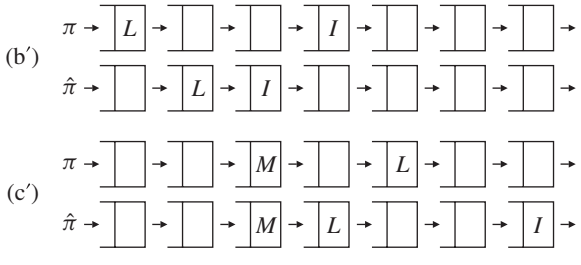

FIGURE 3: Job positions in cases (b′) and (c′).

we continue the argument as before from that point, noting the fact that the flowtime for job J (and possibly for job K or job L if one of them completes its last task) is shorter under $\hat{\pi}$ than under π. Therefore, suppose that job J completes under π and that the job that completes under $\hat{\pi}$ is a job in a strictly earlier buffer than is job I (so $K = I$ and L completes; see diagram (c-ii) of Figure 2 for the job positions just before completion); we call this case (b′). Note that J completes earlier under $\hat{\pi}$ than under π, and that all other job completion times are the same up to this point.

(b′) The relative states for π and $\hat{\pi}$ are again as in case (b), except in reverse: the later job I is one buffer further along under π, and the earlier job L is one buffer further along under $\hat{\pi}$. Also, I has completed the most tasks and L has completed the second-greatest number of tasks under both policies. Thus, upon letting $\hat{\pi}$ agree with π, under both policies job I will be served by worker one and job L will be served by worker two. We continue until either job L passes job I under $\hat{\pi}$ when the states are the same (case (a)), or until job I completes under π (case (c′)).

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

(c′) The relative states are as in case (c), except the roles of π and $\hat{\pi}$ are reversed and the jobs are respectively I and L instead of J and I, where L has completed the most tasks after job I. Thus, under π worker one will serve job L and worker two will serve a job, M, that is in an earlier buffer than is L, while under $\hat{\pi}$ worker one serves job I and worker two serves job L. We do the same coupling of the workers as we did in case (c), reversing the roles of π and $\hat{\pi}$ and with job I taking the role of job J and job L taking the role of job I, until job I completes under $\hat{\pi}$. At this point the states will either be the same (case (a)), or we will be back to case (b) with jobs L and M taking the roles of jobs J and I, and with job L being the job with the most task completions and job M being the job with the second-greatest number of task completions under both policies. Note that job I completes earlier under π than it does under $\hat{\pi}$ in this case.

We keep repeating this coupling until either the time horizon is reached or the states under the two policies become the same. In the meantime, the relative states will alternate between (c) and (c′), with visits to (b) and (b′) in between. Jobs will have different departure times only when the processes go through relative states as in (c) and (c′). Let t_1 be the first time we go to a state as in (c) when I has the second-greatest number of task completions after J. Before that time, we cannot have been in states as in (b′) or (c′), and job departure times before then are earlier under $\hat{\pi}$ than they are under π. After t_1, until either the states become the same (case (a)) or we reach the time horizon, the departure times under π will be $t_1 + \delta_1, t_2, t_3 + \delta_3, t_4, \ldots$, while $\hat{\pi}$ will have corresponding departures at times $t_1, t_2 + \delta_2, t_3, t_4 + \delta_4, \ldots$, where t_2 is the departure time of job I under π, t_3 is the departure time of job L under $\hat{\pi}$, etc., and δ_i are independent and identically exponentially distributed with rate μ_1. Thus, the mean flowtime, or sum of mean departure times, for the first k jobs is smaller under $\hat{\pi}$ than it is under π.

Now suppose that at time 0, under π worker one serves job I, in buffer i, and worker two serves job J, in buffer j, where $i < j$, and no other jobs are in buffers $i + 1, \ldots, n$. Let $\hat{\pi}$ have worker one serve J and worker two serve I. Let us couple the first event such that job I completes under both policies with probability μ_2, job J completes under both policies with probability μ_2, and job J completes under $\hat{\pi}$ and job I completes under π with probability $\mu_1 - \mu_2$. Then, at the time of the first service completion, we are in states as in one of (a), (b), and (c), and the argument proceeds as before.

Now assume that $w \geq 2$ but $\mu_i \equiv \mu$. The argument is the same as above, except for the coupling in cases (c) and (c′). Let us consider case (c), to be specific. Now $w - 1$ of the workers serve the same tasks under each policy, but one worker serves task J under π and task M under $\hat{\pi}$. We couple all the workers directly under the two policies so that the same worker completes under each policy. The rest of the argument is as before. The argument for the case $n = 2$ is also similar to the one above; now there are no cases (b) and (b′). The reason our argument does not work for general w, n, and μ_i is that, in case (c′), for general parameters, it is possible to go from (c′) back to (b′) rather than to (b), and it is therefore possible for π to have two departures in a row that are both earlier than the corresponding departures under $\hat{\pi}$.

We have constructed a policy $\hat{\pi}$ that agrees with LBFS at time 0, and with smaller mean flowtime than π, so the proof is complete.

Note that the LBFS policy can be implemented with a bucket brigade-type policy, where workers are arranged from slowest to fastest, with the fastest given priority. We assume, in contrast to Bartholdi *et al.* [10], that the equipment is such that multiple workers may work at the same station (but not on the same job). Note that under the standard bucket brigade policy, full cross-training is not required: the $(n - k)$th worker can never work at any of the last k stations, and the kth worker can never work at any of the first $k - 1$ stations. Thus, the standard bucket brigade operates under a type of nested zone arrangement.

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

Corollary 4.8. *For exponential fully trained workers and an infinite supply of jobs at the first station or a closed system, if* $w = 2$, $n = 2$, *or* $\mu_i \equiv \mu$, *then a modified bucket brigade, in which multiple workers can work at the same station, minimizes the mean flowtime process.*

Theorem 4.4, though difficult to prove, is quite restrictive: the objective is minimizing the mean flowtime process rather than stochastically minimizing a more general process, an infinite number of jobs at the first station or a closed system is assumed, and workers must either be homogeneous or there can be at most two workers or two tasks. However, the only existing theoretical result for the optimality of the standard bucket brigade is that it achieves the maximal long-run throughput. For that objective, pick-and-run is also optimal, where (recall) by pick-and-run we mean that a worker starts a new job with its first task and then follows that job without interruption until its last task is completed.

Theorem 4.4 also strengthens the result of [28] that, for identical exponential workers, the pick-and-run policy minimizes the mean cycle time.

For systems with only two tasks we can show the following result, which complements a result of Ahn *et al.* [3]. They showed that when there are two identical workers and two tasks, no arrivals, and processing times depend on the tasks and not the workers, LBFS minimizes the total mean holding cost when $h(2)\eta_2 \geq (h(1) - h(2))(\eta_1 + \eta_2)$.

Theorem 4.5. *Assume that there are an infinite number of jobs at the first station, only two tasks, and an arbitrary number of heterogeneous, fully trained workers. Suppose that* $h(2) \geq h(1)/2$. *The nonidling LBFS policy stochastically minimizes the holding cost process (or weighted flowtime),* $\{\mathcal{H}_k\}_{k=0}^{\infty}$, *where* $\mathcal{H}_k$ *is the total holding cost for the first* k *jobs to depart.*

Proof. The initial part of the proof follows the proof of Theorem 4.4, with π serving a 1-task at time 0 using worker S when a 2-task is not served at time 0, and then following LBFS, and with $\hat{\pi}$ serving the 2-task using S. At time 1 either the states are the same (case (a) above) or π has two more 2-tasks and one fewer 1-task than $\hat{\pi}$ (case (c)). Now, for case (c), let $\hat{\pi}$ agree with π from time 1 on, except in that, for two of the 2-tasks that π serves, $\hat{\pi}$ serves two 1-tasks, using the same workers, until one of these two workers completes, at time t, say. At that point the states will be the same (case (a)). Between times 1 and t all other task and job departures will be the same, and the holding cost rate under π will be higher by $2h(2) - h(1)$.

When our objective function is to stochastically maximize the total task completion process (and, thus, also to maximize worker utilization and throughput for stable systems), we can permit open systems with an arbitrary arrival process. The following can be shown with a simpler variant of the proof of Theorem 4.4.

Theorem 4.6. *For an open system with fully trained, randomly unavailable, and noncollaborating workers, the nonidling FBFS policy stochastically maximizes the total task completion process,* $\{\tilde{C}_t^1\}_{t=0}^{\infty}$.

5. Conclusion

In this paper we have considered the problem of dynamic allocation of a flexible workforce in a generalized serial queueing system. Rather than focusing on a complete characterization of the optimal policy for each specific problem instance, our aim was to identify the properties that optimal policies must share for many problems of similar kind. We gave a partial characterization of the optimal policy for very general systems and we further refined the structure of the optimal policy for deterministic and exponential processing times. We identified properties

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

of good policies and the corresponding objective functions for which optimal policies exhibit such properties, thereby significantly reducing the computational effort required to develop good heuristics.

Our main results can be summarized as follows.

1. For fixed tasks, LBFS and FCFS each

   (a) stochastically maximizes the job completion process, $\{c_t^n\}_{t=0}^{\infty}$, with preemption, idling, open or closed systems, and ILR processing times; and

   (b) stochastically maximizes the joint task completion process, $\{\bar{C}_t\}_{t=0}^{\infty}$, and stochastically minimizes the holding cost process, $\{H_t\}_{t=0}^{\infty}$, with no preemption, no idling, open or closed systems, and general independent, identically distributed processing times.

2. For fixed tasks and the last shared task of exclusive workers, and with no collaboration, LBFS stochastically maximizes the joint task completion process, $\{\bar{C}_t\}_{t=0}^{\infty}$, with idling, with open systems, and when processing times are deterministic (no preemption) or exponential (with preemption).

3. For exponential processing times, with idling and preemption, LBFS

   (a) minimizes the mean flowtime process, $\{\mathrm{E}\, F_k\}_{k=0}^{\infty}$, when there are an infinite number of jobs at the first buffer (or the system is closed), workers are fully trained and cannot collaborate, and $w = 2$, $n = 2$, or $\mu_i \equiv \mu$; and

   (b) stochastically maximizes the joint task completion process, $\{\bar{C}_t\}_{t=0}^{\infty}$, with open or closed systems, collaboration, and nested zones.

4. FBFS stochastically maximizes the total task completion process, $\{\bar{C}_t^1\}_{t=0}^{\infty}$,

   (a) with open systems, fully trained, randomly unavailable workers, and unit processing times; and

   (b) with open systems, fully trained, randomly unavailable, noncollaborating workers, and exponential processing times.

Acknowledgements

We wish to thank Gido Brouns for helpful discussions, Dimitrios Pandelis for pointing out an error in a previous version, and the referee for careful and constructive feedback.

References

[1] Ahn, H.-S. and Righter, R. (2005). Multi-actor Markov decision processes. *J. Appl. Prob.* **42,** 15–26.
[2] Ahn, H.-S., Duenyas, I. and Lewis, M. E. (2002). The optimal control of a two-stage tandem queueing system with flexible servers. *Prob. Eng. Inf. Sci.* **16,** 453–469.
[3] Ahn, H.-S., Duenyas, I. and Zhang, R. Q. (1999). Optimal stochastic scheduling of a 2-stage tandem queue with parallel servers. *Adv. Appl. Prob.* **31,** 1095–1117.
[4] Andradóttir, S., Ayhan, H. and Down, D. G. (2001). Server assignment policies for maximizing the steady-state throughput. *Manag. Sci.* **47,** 1421–1439.
[5] Andradóttir, S., Ayhan, H. and Down, D. G. (2003). Dynamic server allocation for queueing networks with flexible servers. *Operat. Res.* **51,** 952–968.
[6] Askin, R. G. and Chen, J. (2006). Dynamic task assignment for throughput maximization with worksharing. *Europ. J. Operat. Res.* **168,** 853–869.
[7] Askin, R. G. and Chen, J. (2006). Throughput maximization in serial production lines with worksharing. *Internat. J. Production Econom.* **99,** 88–101.

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press

[8] Asmussen, S. and Koole, G. M. (1993). Marked point processes as limits of Markovian arrival streams. *J. Appl. Prob.* **30,** 365–372.
[9] Bartholdi III, J. J. and Eisenstein, D. D. (1996). A production line that balances itself. *Operat. Res.* **44,** 21–34.
[10] Bartholdi III, J. J., Eisenstein, D. D. and Foley, R. D. (2001). Performance of bucket brigades when work is stochastic. *Operat. Res.* **49,** 710–719.
[11] Bischak, D. P. (1996). Performance of a manufacturing module with moving workers. *IIE Trans.* **28,** 723–733.
[12] Duenyas, I., Gupta, D. and Olsen, T. L. (1998). Control of a single server tandem queueing system with setups. *Operat. Res.* **46,** 218–230.
[13] Farrar, T. (1993). Optimal use of an extra server in a two station tandem queueing network. *IEEE Trans. Automatic Control* **38,** 1296–1299.
[14] Gel, E. S., Hopp, W. J. and van Oyen, M. P. (2000). Factors affecting opportunity of worksharing as a dynamic line balancing mechanism. *IIE Trans.* **34,** 847–863.
[15] Gel, E. S., Hopp, W. J. and van Oyen, M. P. (2006). Opportunity of hierarchical cross training in serial production. To appear in *IIE Trans.*
[16] Grassman, W. K. and Tavakoli, J. (2002). A tandem queue with a movable server: an eigenvalue approach. *SIAM J. Matrix Anal. Appl.* **24,** 465–474.
[17] Hopp, W. J., Tekin, E. and van Oyen, M. P. (2004). Benefits of skill chaining in serial production lines with cross-trained workers. *Manag. Sci.* **50,** 83–98.
[18] Iravani, S. M. R., Posner, M. J. M. and Buzacott, J. A. (1997). Two-stage tandem queue attended by a moving server with holding and switching costs; static and semi-dynamic policy. *Queueing Systems* **26,** 203–228.
[19] Javidi, T., Song, N.-O. and Teneketzis, D. (2001). Expected makespan minimization on identical machines in two interconnected queues. *Prob. Eng. Inf. Sci.* **15,** 409–444.
[20] Koole, G. and Righter, R. (1998). Optimal control of tandem reentrant queues. *Queueing Systems* **28,** 337–347.
[21] Mandelbaum, A. and Reiman, M. I. (1998). On pooling in queueing networks. *Manag. Sci.* **44,** 971–981.
[22] McClain, J. O., Thomas, L. J. and Sox, C. (1992). 'On-the-fly' line balancing with very little WIP. *Internat. J. Production Econom.* **27,** 283–289.
[23] Ostolaza, J., McClain, J. and Thomas, J. (1990). The use of dynamic (state-dependent) assembly-line balancing to improve throughput. *J. Manufacturing Operat. Manag.* **3,** 105–133.
[24] Pandelis, D. G. (2006). Optimal use of excess capacity in two interconnected queues. To appear in *Z. Operat. Res.*
[25] Pandelis, D. G. (2006). Optimal control of flexible servers in two tandem queues with operating costs. Preprint.
[26] Pandelis, D. G. and Teneketzis, A. D. (1994). Optimal multiserver stochastic scheduling of two interconnected priority queues. *Adv. Appl. Prob.* **26,** 258–279.
[27] Sennott, L., van Oyen, M. P. and Iravani, S. M. R. (2006). Optimal dynamic assignment of a flexible worker on an open production line with specialists. *Europ. J. Operat. Res.* **170,** 541–566.
[28] Van Oyen, M. P., Gel, E. G. S. and Hopp, W. J. (2001). Performance opportunity for workforce agility in collaborative and noncollaborative work systems. *IIE Trans.* **33,** 761–777.

https://doi.org/10.1239/aap/1158684995 Published online by Cambridge University Press