Manufacturing & Service Operations Management

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Properties of Optimal-Weighted Flowtime Policies with a Makespan Constraint and Set-up Times

Mark P. Van Oyen, Juta Pichitlamken,

To cite this article:
Mark P. Van Oyen, Juta Pichitlamken, (2000) Properties of Optimal-Weighted Flowtime Policies with a Makespan Constraint and Set-up Times. Manufacturing & Service Operations Management 2(1):84-99. http://dx.doi.org/10.1287/msom.2.1.84.23264

Properties of Optimal-Weighted Flowtime Policies with a Makespan Constraint and Set-up Times

Mark P. Van Oyen • Juta Pichitlamken
Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois 60208-3119

We characterize optimal policies for the problem of allocating a single server to a set of jobs from N families. Each job is an instance of demand for an item and is associated with a family, a holding cost rate, and a mean processing time. Set-up times are required to switch from one family to another, but are not required to switch within a family. We consider the case in which the order of jobs within the family is unconstrained, and a variation in which the order is fixed. The optimization is with respect to the weighted flowtime, and we treat problems both with and without a makespan-constraint. Practical examples based on this model are described. We partially characterize an optimal policy by means of a Gittins reward-rate index and a similar switching index derived from multi-armed bandit theory. For deterministic problems with a makespan constraint, we present an optimization algorithm for the special case of two families and at most three set-ups . Without a makespan constraint and without preemption, we prove that our analysis of a deterministic model extends to stochastic set-up and processing times without loss of optimality. Managerial insights based on our technical results are provided.
(*Job Scheduling; Job Families; Multi-armed Bandit; Group Technology; Weighted Flowtime; Makespan; Set-up Time*)

1. Introduction

We consider a single machine in a mixed-model production system. The models are grouped into N job/model families (alternately, parallel queues). Each job is identified by a vector (f, s), where f designates the family to which the job belongs and s indicates that the job is of item type s within family f. The machine requires a significant set-up time to switch from one job family to another, but no set-up time is required within a job family. Items within a family are further differentiated by their holding costs and average processing times. There are no arrivals: we are concerned with the scheduling of the jobs initially present in the system. We seek a nonpreemptive scheduling policy which minimizes the total holding cost incurred by unprocessed jobs in the system. Set-up times provide an incentive to batch the service of jobs belonging to the same family, and make it difficult to construct an optimal policy. We extend our deterministic scheduling model to include a makespan constraint. The intent of this paper is to systematically expose the significant structure inherent in this model and to draw insights from it.

As an illustration of how our model can be employed in practice, consider a manufacturing plant that employs a flexible manufacturing system to stamp metal products, which are then shipped to several assembly plants. At this manufacturing facility, a single, large punch-press machine is used to execute a series of three to eight stamping operations to produce the

1523-4614/00/0201/0084$05.00
1526-5498 electronic ISSN

metal product from rolls of metal sheet. There are N product families, distinguished by the die that is used to stamp parts of that family. Within a family, there are a number of items that differ with respect to the particular stamping operations, but are formed with the same die. Because the same press produces all of the product families, a set-up time is required to change production from one family to another. The dies are expensive and complex, and the set-up time for changing a die is on the order of 4–8 hours. The manufacturing plant operates in a make-to-stock (MTS) mode. Orders are sent to the manufacturing plant, and inventory is replenished on a weekly basis, based on the finished goods inventories at the assembly plants. Whenever a reorder point is hit at one of the assembly plants, it places an order for $Q_{f,s}$ parts with job vector (f, s). Our scheduling model is based on the mean processing times and inventory holding costs for each item and the mean set-up time for each family. Holding costs can be used to assign relative importance across part families and even within a family as demonstrated in the following two scenarios.

In the first scenario, consider that two assembly plants may carry the same part; but the first plant's order is placed simply to replenish its spare-parts inventory, while the second plant is requesting the parts on backorder. In such a case, two items may be created for the same part at the manufacturing plant: one with a small holding cost for the replacement parts ordered by the first plant, and the second with a relatively large holding cost to account for the urgency of the second plant's backordered parts.

In the second scenario, suppose that some parts are purchased by multiple assembly plants so that the production system will at times receive multiple requests for the same family and item. In operating the MTS production system, the reorder quantities and base-stock levels must be set appropriately to provide a sufficient service level (e.g. fill rate). To gain system efficiency by combining orders that share set-ups, orders are accumulated over a period of time, H, which we take to be one week. After H time units, the orders are sequenced on the machine. It is important that the scheduling policy be efficient and also conclude the final order within H time units, thereby ensuring the stability of the system. Effective schedules should place a higher priority on filling the orders that have been received earlier (since they face a greater likelihood of stocking out prior to resupply). For instance, suppose one wishes to make an H-day old order for job vector (f, s) about twice as costly to delay as a one-day-old order. This can be done by setting the inventory holding cost of part family f and item s as $c_{f,s}$ where $c_{f,s} = 2H - 1 - d_{f,s}$ and using a makespan constraint of H days, where $d_{f,s} \in \{1,2,\ldots,H\}$ is the day of the cycle on which the order arrived. In this way, the scheduling task has an incentive to complete earlier orders sooner. Assuming $H = 7$, a task arriving on the first day of the cycle is assigned a holding cost of 12, while a job order on day seven receives a holding cost of six. The makespan constraint would be set at seven days as well. As we shall see, set-up times make it difficult to determine which overall production schedule is optimal. It is certain from the structure of an optimal policy, however, that orders arriving earlier receive higher indices and thus will necessarily be processed sooner (within the same family). This scenario illustrates an important theme which has been neglected in prior literature on this type of model: that it is important to consider the impact of a makespan constraint on the weekly production schedule.

A Group-Technology (GT) policy is one that serves all the jobs of a given family consecutively without an interruption. In this way, it generates a schedule with a minimal feasible makespan, because each family is set up only once. The slack time between the end of the scheduling period, H, and this GT makespan indicates the freedom allowed in searching for higher-performance policies that set up at least one family more than once. One of our main results identifies index-based stopping times that characterize an optimal policy in both unconstrained systems and makespan-constrained systems.

We examine two variations of our sequencing model: one static (**S**) and the other dynamic (**D**). In the static problem, the job sequence within the same family is fixed, while we allow the control policy to choose the job sequence in the dynamic case. The formulation and analysis of the static sequencing problem, **S**, is a novel feature of this paper. Systems can be modeled as problem **S** when there are independently competing

projects (families), but precedences on the project subtasks bind the sequence within a project. In particular, the sequence may be fixed to coincide with the earliest due date (EDD) ordering or FCFS order of job arrivals prior to the start of the production cycle. Moreover, if one wishes to exclude holding costs altogether by setting them all equal to one, the only way to introduce priorities is through a job ordering such as we have in **S**. For both problems **S** and **D** without the makespan constraint, we show that an optimal dynamic scheduling policy can be obtained without loss of optimality by analyzing a deterministic version based upon mean values only.

We also introduce the novel feature of a makespan constraint in the deterministic setting. It is important that strong structural properties of an optimal policy can be derived for such systems, because makespan-constrained models are helpful in bridging the analysis of models without arrivals to practical scheduling algorithms in applications with arrivals. Fully dynamic policies for stochastic models with a makespan constraint are highly complex, since one must quantify the probability of meeting the constraint and finite end-of-horizon effects. For deterministic models, however, we present a unified analysis yielding new insights into the structural properties of optimal policies of both problems **S** and **D** with and without a makespan constraint. We present an optimization algorithm for the special case of two families and at most three set-ups.

The scheduling model treated here captures the issues of when to set up a new family and how to sequence various items *within the family* currently set up. Thus, the key issues include: (1) sequencing jobs within a "batch" belonging to the same family (problem **D** only), (2) determining batch sizes, and (3) sequencing batches associated with distinct families. We partially characterize an optimal policy by means of a Gittins reward-rate index and a similar switching index derived from the multi-armed bandit theory. In this way the set of possible batches is limited to a set of candidates defined by "switching indices." This paper is the first to develop reward-rate based switching indices appropriate to problems with job families. All of our results for unconstrained optimization apply to problems with a makespan constraint as well. For makespan-constrained problems, we present an optimization algorithm for the special case of two families and at most one additional set-up.

This paper is organized as follows: We begin with a survey of the most relevant literature and describe our contribution in §2. The problem formulation is described in §3. Section 4 contains definitions of reward rate, switching index, and stopping times; an analysis of the structural properties of the optimal policies, and sufficient conditions for optimality. In §5, the reward rate and stopping time concepts are used to address the weighted flow time problem with a makespan constraint. We summarize our conclusions and managerial insights in §6. A glossary of notation is listed in Appendix A, and the proofs of some results are shown in Appendix B for clarity.

2. Literature Survey

The literature is large for the problem at hand. A partial list includes Ahn and Hyun (1990), Bruno and Sethi (1987), Ham et al. (1985), Mason and Anderson (1991), Monma and Potts (1989), Potts and Wassenhove (1992), Van Oyen et al. (1998) and (1992), and Webster and Baker (1994). See Ahn and Hyun (1990) for a discussion of how such models arise in steel-pipe making: families "may be based on the outside diameters, and furthermore, the jobs in the same class can be produced with the same roller (family scheduling or group technology concept), even though their inside diameters, lengths and values may differ."

The special case with only one item per family has been completely characterized (see Santos and Magazine (1985) for deterministic models and Van Oyen et al. (1992) for a stochastic formulation and analysis). With D_n denoting the job set-up time, c_n the holding cost rate per job, μ_n^{-1} the mean service time, and x_n the number of identical jobs of family n, the "index of family n" is given by $\bar{\nu}_n = c_n\mu_n(x_n\mu_n^{-1})/(x_n\mu_n^{-1} + E\{D_n\})$. This rule can be interpreted as the familiar $c\mu$ index multiplied by the fraction of time over which useful work is performed. These results are extended to queues interconnected by job feedback in Van Oyen and Teneketzis (1994).

Using a deterministic framework, Baker (1993) explores and surveys problems in which jobs may be organized according to families. We refer to the restriction to a policy that serves all of the jobs of a given family consecutively without an interruption as the *GT assumption*. This assumption is appropriate when set-up times or the nature of the production system require that all jobs of the same family be served together in a batch. See Potts and Van Wassenhove (1992) for a taxonomy and literature survey emphasizing complexity analysis. Ham, Hitomi, and Yoshida (1985) first showed that the optimal ordering of the batches can be obtained as a sequence in nonincreasing order of the index $\hat{C}(f)/[D_f + \hat{\mu}(f)^{-1}]$, where $\hat{C}(f)$ is the sum of all holding costs for the jobs of family f and $D_f + \hat{\mu}(f)^{-1}$ is the total time required to set-up and to serve family f (see Theorem 7.5). The stochastic version of this GT problem was briefly treated in Van Oyen et al. (1998). It is perhaps surprising that the "index of family n" is the same with or without the GT assumption and does not depend on the particular ordering of jobs within the batch (compare Van Oyen et al. (1998) and (1992)).

Asawa and Teneketzis (1996) developed an innovative perspective on the characterization of classical multi-armed bandits (with or without discounting) under the additional (nonclassical) feature of lump-sum switching costs or switching delays (times). Although different in detail, the concepts from Van Oyen et al. (1992) prove applicable here and we draw upon these and other concepts to provide a relatively strong characterization of an optimal policy.

3. Problem Formulation

Although the literature typically casts the scheduling model as intrinsically deterministic, we show that when there is no makespan constraint, the stochastic scheduling problem can be analyzed via its deterministic counterpart using mean values. For this reason, we proceed with a stochastic formulation of the scheduling problem.

A single server is to be allocated to an arbitrary number of jobs, all initially available for processing (zero release times). Each job belongs to a job family, and the families are labeled 1, 2, . . . , N, where $N \in \mathbb{N}$ (where $\mathbf{N}$ denotes the naturals and $\mathbb{Z}^+$ the nonnegative integers). Family f has $N_f \in \mathbb{N}$ items, where we designate a job by the vector (f, s) if it belongs to family f and item s. Let $n(f, s)$ denote the number of (identical) jobs with the job vector (f, s). Service times for job vector (f, s) have i.i.d. service times $\sigma(f, s, \cdot)$ drawn from a distribution $B(f, s)$ with mean $\mu(f, s)^{-1} \in (0, \infty)$. Any job with vector (f, s) incurs a holding cost weight of $c(f, s) \in (0, \infty)$ per unit time it spends in the system.

Set-ups are required to switch production from a job of one *family* to another. There is no switching penalty for changes from one item to another within the same family. The set-up time for family f is $\delta(f)$, which is drawn from the distribution $\Delta(f)$ independently of all else and has mean $D_f \in [0, \infty)$.

We begin by considering the case without a makespan constraint. The objective is to determine a (dynamic) scheduling policy that minimizes the total infinite horizon expected holding cost (equivalently, weighted flowtime). We restrict attention to nonpreemptive and nonanticipative policies. That is, services and set-ups must be completed once initiated; the server is active while jobs are present; and service/set-up times may not be used by the policy in advance of their realization. It is straightforward to prove that (1) idling policies and (2) policies that have consecutive set-ups (that is, one set-up followed by a second one that makes the first unnecessary) are strictly suboptimal. Thus, we may restrict attention to policies that serve at least one job of family f following a set-up of f. A policy specifies, at each job completion epoch, that the server either remain working in the present family on a particular job, or set up another job family for service. At time zero, we assume that no family is set up, and therefore a set-up is required.

We examine two variations on the problem at hand, one static and the other dynamic.

- **Static Sequence Problem (S):** In the static case, the job sequence within a family is fixed.
- **Dynamic Sequence Problem (D):** In the dynamic case, we allow the control policy to choose the job sequence within a family in addition to controlling the times at which the policy will switch from one family to another.

Under the assumptions of the formulation, which include nonpreemptive services and set-ups, even the fully dynamic scheduling problem reduces to a deterministic problem (we refer the reader to Appendix B

for the details). For the sake of streamlining our notation, we hide the set-ups by defining a new job vector as follows: We take $(f(i), 0)$ to represent a set-up of family $f(i)$ and thus $c(f(i), 0) \triangleq 0$, $\sigma(f, 0) \triangleq \delta(f)$, and $\mu(f, 0)^{-1} \triangleq D_f$. Let g_i designate the job vector $(f(i), s(i))$ served on the ith stage of a policy g. We can represent $g \in G^L$, where G^L is the space of list (open-loop) policies, as a string (list)

$$g = g_1\, g_2\, g_3 \cdots g_{k(g)}, \tag{3.1}$$

where $k(g)$ denotes the number of jobs initially in the system plus the number of set-ups dictated by g. Define for job i of policy g

$$c_i \triangleq c(f(i), s(i)), \tag{3.2}$$

$$\mu_i \triangleq \mu(f(i), s(i)). \tag{3.3}$$

With $f_g(i)$ defined as the flowtime of the ith job under policy g, we can now conveniently express the cost of a policy $g \in G^L$ as

$$F_w(g) = E\left\{\sum_{i=1}^{k(g)} c_i f_g(i)\right\} \tag{3.4}$$

$$= \sum_{i=1}^{k(g)} c(f(i), s(i))\left\{\sum_{k=1}^{i} \mu(f(k), s(k))^{-1} \mathbf{I}\{s(k) \neq 0\} + D(f(k))\mathbf{I}\{s(k) = 0\}\right\}$$

$$= \sum_{i=1}^{k(g)} c_i\left\{\sum_{k=1}^{i} \mu_k^{-1}\right\}, \tag{3.5}$$

where set-up times are captured via Equation (3.2) and Equation (3.3) and **I** denotes the indicator function.

At least one optimal policy exists, because the policy space is finite. Moreover, since a policy is optimal if it minimizes $F_w(g)$ as defined in Equation (3.5), we observe that the solution of this stochastic scheduling problem is identical to the deterministic scheduling problem with constant service times $\mu(f, s)^{-1}$ and set-up times D_f. This simplifies the analysis considerably, and we proceed to analyze structural properties of the original system for problems **S** and **D** using its deterministic counterpart.

After treating the above problem as a deterministic problem, we also consider in §5 the deterministic problem of either type **S** or **D** with the objective of minimizing weighted flowtime subject to a constraint, T, on the schedule's makespan.

We have chosen not to model tear-down times following the service of a batch of family f. This is not restrictive. Mason and Anderson (1991) show in Theorem (1) that a model with mean set-up time D_f and mean tear-down time T_f has the same optimal policy as a model with $D_f' = D_f + T_f$ and $T_f = 0$. In their paper, further generalizations are shown as well.

4. Analysis

4.1. Definitions of Reward Rate, Switching Index, and Stopping Time

Define a (familial) batch to be a string of jobs of family f, denoted B_f, such that $g = M'B_fM''$ for some prefix string M', some suffix string M''; and B_f represents a batch of jobs of family f. Thus, the first stage of B_f is the set-up $(f, 0)$. B_f contains no other set-ups, and M'' begins with a set-up of another family $f'' \neq f$.

For an initial string $g = g_1\, g_2\, g_3 \cdots g_{k(g)}$, the reward rate for any substring $M = g_n\, g_{n+1} \cdots g_{n+m}$ is defined by

$$\bar{\nu}(M) \triangleq C(M)\, \mathcal{M}(M) \tag{4.1}$$

$$C(M) \triangleq \sum_{i=n}^{n+m} c_i, \tag{4.2}$$

$$\mathcal{M}(M) \triangleq \left[\sum_{i=n}^{n+m} \mu_i^{-1}\right]^{-1}. \tag{4.3}$$

Notice that $\bar{\nu}[(f, i)] = c(f, i)\mu(f, i)$, which can be regarded as the (reward) rate at which holding costs are reduced by serving a job (f, i) (see Duenyas and Van Oyen (1996) for elaboration).

Consider now the ordering of batches. Let $g = M'B_fB_hM''$, where B_f and B_h are batches with $f \neq h$ and n_f and n_h stages respectively, and where M' (M'') does not end (begin) with a job of family f (h). Then, letting stage $n' + 1$ of policy g denote the first stage of B_f, we can use Equation (4.1) to calculate the reward rates for batch B_f (and similarly for B_h) as

$$\bar{\nu}(B_f) \triangleq \sum_{i=n'+1}^{n'+n_f} c(f, s(i)) \Big/ \left[\sum_{i=n'+1}^{n'+n_f} \mu(f, s(i))^{-1}\right] = C(B_f)\mathcal{M}(B_f). \quad (4.4)$$

We then define the switching index and the stopping time. Suppose that a partial schedule has been generated, and $(f, 1), (f, 2), \ldots, (f, n-1)$ with $n \leq N_f$ have been served some time prior to t, but family f may not be set up at the current decision epoch t. Using the terminology of Ishikida and Varaiya (1994), we define at any time t, the *load level* of family f to be the next job waiting for processing (n in this scenario). We wish to compute the maximum reward rate that can be earned by serving available jobs of family f, beginning at load level n. Two cases must be considered: first, the case where a set-up is required for family f before job (f, n) can be processed; and secondly, the case where family f is already set up. In problem **D**, the second case is trivial because $c(f, n)\mu(f, n)$ maximizes the reward rate achievable by serving jobs of family f. In problem **S**, the reward rate of the second case is known to be the Gittins dynamic allocation index (DAI). Thus, it only remains to consider the case where family f requires a set-up. We define the switching index as $\nu_{f,n}(1)$, the maximum reward rate available upon switching to family f at load level n, and the corresponding stopping time, $\tau_{f,n}(1)$, as follows. Because an optimal policy may continue to serve family f beyond $\tau_{f,n}(1)$, it is useful to define $\nu_{f,n}(j)$ for $j = 1, 2, \ldots$, which is the jth best reward rate achievable, and the corresponding stopping time, $\tau_{f,n}(j)$. Let $\tau_{f,n}(0) \triangleq n - 1$, then recursively compute until $\tau_{f,n}(j) = N_f$:

$$\nu_{f,n}(j) \triangleq \max\{\bar{\nu}[(f, 0)\ (f, n) \cdots (f, \tau)];\ \tau_{f,n}(j-1) < \tau \leq N_f\} \quad (4.5)$$

$$\tau_{f,n}(j) \triangleq \max\{\tau : \bar{\nu}[(f, 0)\ (f, n) \cdots (f, \tau)] = \nu_{f,n}(j);\ \tau \leq N_f\} \quad (4.6)$$

for $k > 0$, $\bar{\nu}[(f, 0)\ (f, n) \cdots (f, N_f + k)] \triangleq 0$. Thus, $\tau_{f,n}(1)$ is the *longest* stopping time that achieves $\nu_{f,n}(1)$. Provided the following quantities exist for $j, k \in \mathbb{N}$ Equations (4.6) and (4.5) imply the following intuitive structure:

$$\nu_{f,n}(j) > \nu_{f,n}(j+k), \quad (4.7)$$

$$\nu_{f,n}(j+k) > \bar{\nu}[(f, 0)(f, \tau_{f,n}(j)+1) \cdots (f, \tau_{f,n}(j+k))]. \quad (4.8)$$

Note also that the right hand side of Equation (4.8) is not necessarily equal to $\nu_{f,\tau_{f,n}(j)+1}(k)$, because the job services involved are crucial in selecting the stopping time. The definition (4.5) is equivalent to the *switching reward rate index* defined for a class of discrete time bandit processes in Asawa and Teneketzis (1996). Moreover, this definition helps reduce the number of potentially optimal policies; see Corollary 4.3 of Mason and Anderson (1991).

4.2. Structural Properties

We begin with the property that all jobs sharing the same family and item designation may be served consecutively. That is reasonable because jobs of the same vector are identical and thus share a common incentive for serving them, thereby giving each the same "priority". This is a very intuitive result, but also a very practical one. In production applications, it is common to have only a small number of product models (items) and a small number of priorities within each model. Lemma (1) offers a tremendous reduction in complexity by collapsing (in **D**) all of the jobs of the same item into a single job. Even in problem **S**, the reduction applies to consecutive jobs of the same item.

LEMMA 1. *For problems* **S** *and* **D** *there exists an optimal policy that serves all the jobs with a common vector (f, s) exhaustively. That is, it is optimal to consecutively serve all n(f, s) jobs with job vector (f, s).*

The proof can be found in Appendix B.

Because all jobs of the same family and item may be served consecutively under an optimal policy, the $n(f, s)$ jobs with vector (f, s) are served consecutively in a period of length $n(f, s)\,\mu(f, s)^{-1}$. Thus, for the purpose of optimal scheduling, it suffices to consider a problem, say P', that is equivalent to the original problem, except for the following parameters: $n'(f, s) = 1$, $c'(f, s) = n(f, s)c(f, s)$, and $\mu'(f, s)^{-1} = n(f, s)\,\mu(f, s)^{-1}$. To see this, consider any two scheduling policies g and $\tilde{g}$. Let $r \triangleq F_w(g) - F_w(\tilde{g})$, as computed under the original formulation, P. Note that under P', $F_w^{P'}(g) - F_w^{P'}(\tilde{g}) = r$ as well. For any g, the effect of P' is a constant:

$$F_w^{P'}(g) - F_w(g) = \sum_{f=1}^{N} \sum_{s=1}^{N_f} c(f, s) \sum_{i=1}^{(n(f,s)-1)} i\mu(f, s)^{-1}.$$

This is summed up in the following proposition:

PROPOSITION 1. *For the purpose of determining an optimal policy, in problem* **D**, *the n(f, s) jobs with vector (f, s) may be replaced by a single job with holding cost n(f, s)c(f, s) and service rate μ(f, s)/n(f, s).*

Proposition (1) is a slightly more restrictive statement than Corollary (4.3) of Mason and Anderson (1991); however, our proof is different and the result can be extended to problem **S**. In problem **S**, whenever the sequence of family f contains m consecutive jobs with the job vector (f, s), these jobs may be replaced by such a single job as described in Proposition (1).

THEOREM 1. *If policy $g = M'B_fB_hM''$ as defined in § 4.1. is optimal in problem* **S** *or in problem* **D**, *then $\bar{\nu}(B_f) \geq \bar{\nu}(B_h)$.*

PROOF. Suppose that $\bar{\nu}(B_h) > \bar{\nu}(B_f)$. Let $\tilde{g}$ be identical to g except for the interchange of the batch B_f and B_h. It is possible that this interchange results in the savings of either one or two set-ups. We get

$$F_w(g) - F_w(\tilde{g}) \geq C(B_h)\mathcal{M}(B_f)^{-1} - C(B_f)\mathcal{M}(B_h)^{-1}$$
$$= \{\mathcal{M}(B_f)\mathcal{M}(B_h)\}^{-1} [\bar{\nu}(B_h) - \bar{\nu}(B_f)] > 0,$$

where the inequality is strict if one or two set-ups are saved by the interchange of B_f and B_h. □

We wish to emphasize that Theorem (1) leads to the following property of decreasing batch reward rates for batches of the same family. Moreover, in the context of problem **D**, Theorem (1) is shown as Theorem (3) of Mason and Anderson (1991).

COROLLARY 1. *For problems* **S** *and* **D**, *if an optimal policy is of the form $g = B_{f(1)}B_{f(2)} \cdots B_{f(n(g))}$, where batches $B_{f(i)}$ and $B_{f(I)}$ such that $i < I$ both serve the same family, then $\bar{\nu}(B_{f(i)}) \geq \bar{\nu}(B_{f(I)})$.*

The following result, proved as Theorem (1) part (b) of Monma and Potts (1989) and in Bruno and Sethi (1987) indicates that there exists an optimal policy which follows the $c\mu$ ordering in sequencing the items within a family, regardless of how the batches are formed.

THEOREM 2. *Within a family (possibly split into multiple batches), it is optimal under problem* **D** *to process jobs according to nonincreasing value of $c(f, \cdot)\mu(f, \cdot)$.*

Although problem **S** cannot be simplified in this way, in the sequel, we always assume for problem **D** that for each family f, the items are labeled in order of decreasing reward rate. In other words, for each family f, relabel the items such that for $s = 1, \ldots, N_f - 1$, $c(f, s)\mu(f, s) \geq c(f, s + 1)\mu(f, s + 1)$. If problem **D** is without set-ups, an optimal scheduling policy is specified by Theorems (1) and (2). Simply order all jobs according to nonincreasing $c(\cdot, \cdot)\mu(\cdot, \cdot)$ indices. With set-ups, however, an incentive is provided to serve batches of jobs of the same family, because additional holding cost penalties are incurred during set-ups for all jobs remaining in the system. The set-up time affects the resource allocation decision in a complex way that depends on the holding costs and service times of the jobs considered, as well as the set-up time for the family.

By Theorem (2), an optimal policy sequences jobs of a given family in the order 1, 2, ... By Proposition (1), we restrict attention to $n(f, s) = 1$ for all f, s. Having defined the sequence within a batch, Problem **S** and Problem **D** are alike in scheduling the families. The remaining question is how to split each familial string $(f, 1)(f, 2) \cdots (f, N_f)$, into batches and how to splice the various batches together to form an optimal sequence. That is, g, as defined in String (3.1), is composed of a number, say $n(g)$, of successive batches according to $g = B_{f(1)}B_{f(2)} \cdots B_{f(n(g))}$, where each $B_{f(i)}$ is a batch of the form

$$B_{f(i)} = (f(i), 0) (f(i), s(i)) (f(i), s(i) + 1) \cdots (f(i), \tau(i)), \quad (4.9)$$

where $f(i) \neq f(i + 1)$ for $i = 1 \ldots n(g) - 1$, and $\tau(i) - s(i) + 1$ indicates the number of jobs of family $f(i)$ served on the ith batch.

Our use of the notion of reward rate is clarified by the following property, presented in Lemma (2). The following is a standard result that is easily verified (see Corollary (2.1) of Mason and Anderson (1991) for a very similar result).

LEMMA 2. *For any two strings M' and M'' such that $M' = g_1 g_2 \ldots g_{n'}$ and $M'' = g_{n'+1} g_{n'+2} \ldots g_{n'+m'}$ for some n' and m',*

$$\bar{\nu}[M'M''] = \frac{\mathcal{M}(M')^{-1}\ \bar{\nu}(M') + \mathcal{M}(M'')^{-1}\ \bar{\nu}(M'')}{\mathcal{M}(M')^{-1} + \mathcal{M}(M'')^{-1}}. \quad (4.10)$$

$$\min\{\bar{\nu}(M'),\ \bar{\nu}(M'')\} \le \bar{\nu}[M'M''] \le \max\{\bar{\nu}(M'),\ \bar{\nu}(M'')\}. \quad (4.11)$$

Since Equation (4.10) indicates that the reward rate of the concatenation of two strings is a weighted average of the reward rates of the constituent strings, we see that $\bar{\nu}(M'M'')$ is a convex combination of $c_1\mu_1$, $c_2\mu_2, \ldots, c_{n'+m'}\mu_{n'+m'}$. This interpretation of reward rate helps in proving the following lemma which describes a basic property of the switching index and its relation to the stopping time that achieves it. To interpret it, consider $F'F''$ as the continuation of family f from stopping candidate $j-1$ to j. See Appendix B for its proof.

Lemma 3. *For any two strings of family f for $f \in \{1, \ldots, N\}$ such that*

$$F' = (f, \tau_{f,n}(j-1)+1) \cdots (f, \tau_{f,n}(j) - \ell)$$
$$F'' = (f, \tau_{f,n}(j) - \ell + 1) \cdots (f, \tau_{f,n}(j))$$

for some $j, l \in \mathbb{N}$, we have $\bar{\nu}[F'] \le \bar{\nu}[F'']$.

Theorem 3. For problems **S** and **D**, an optimal policy is of the form $g = B_{f(1)}B_{f(2)} \cdots B_{f(n(g))}$, where each $B_{f(i)}$ is a batch of the form $B_{f(i)} = (f(i), 0)\ (f(i), s(i))\ (f(i), s(i)+1) \cdots (f(i), \tau(i))$ such that $\tau(i) \in \{\tau_{f(i),s(i)}(j) : j \in \mathbb{Z}^+\}$. That is, for each i, if batch i begins with job $(f(i), s(i))$, then there exists an optimal policy that stops at one of the candidate stopping times $\tau_{f(i),s(i)}(j)$ for some $j \in \mathbb{N}$.

Proof. Suppose that policy g begins service according to some string M_0 then serves the job string F_0F_1', where

$$F_0 = (f, 0)(f, n) \cdots (f, \tau_{f,n}(j-1)),$$
$$F_1' = (f, \tau_{f,n}(j-1)+1) \cdots (f, \tau_{f,n}(j) - \ell)$$

for some $j, \ell \in \mathbb{N}$. Let M denote the strings of jobs (none of which are of family f) between job $(f, \tau_{f,n}(j) - \ell)$ and batch $(f, 0)\ F_1''\ F_2$, where

$$F_1'' = (f, \tau_{f,n}(j) - \ell + 1) \cdots (f, \tau_{f,n}(j)),$$
$$F_2 = (f, \tau_{f,n}(j)+1) \cdots (f, \tau_{f,n}(j) + \tau')$$

for some $\tau' > 0$. For brevity, we describe g using the string $M_0\ F_0\ F_1'\ M\ (f, 0)\ F_1''\ F_2\ M_1$. We will show that policy g can be improved by serving jobs of family f through to one of the stopping times defined in Equation (4.6). We consider two cases: $\bar{\nu}[F_1''] \ge \bar{\nu}[M(f, 0)]$ and $\bar{\nu}[F_1''] < \bar{\nu}[M(f, 0)]$.

Case 1. If $\bar{\nu}[F_1''] \ge \bar{\nu}[M(f, 0)]$, then the new policy $\tilde{g}$ can be constructed by joining the job string F_1'' with F_1'. Thus, $\tilde{g}$ can be described as $M_0\ F_0\ F_1'\ F_1''\ M\ (f, 0)\ F_2\ M_1$. The cost of the two policies can be compared as follows:

$$F_w(g) - F_w(\tilde{g}) = C(F_1'')\mathcal{M}(M(f, 0))^{-1} - C(M(f, 0))\mathcal{M}(F_1'')^{-1} = \mathcal{M}(M\ (f, 0))^{-1}\ \mathcal{M}(F_1'')^{-1}\ (\bar{\nu}[F_1''] - \bar{\nu}[M\ (f, 0)]) \quad (4.12)$$

Therefore, $F_w(g) - F_w(\tilde{g}) \ge 0$, and policy $\tilde{g}$ is at least as good as g.

Case 2. If $\bar{\nu}[F_1''] < \bar{\nu}[M(f, 0)]$, then the new policy $\hat{g}$ can be constructed by delaying the job of string F_1' until after M. Thus, $\hat{g}$ can be described as $M_0\ F_0\ M(f, 0)\ F_1'\ F_1''\ F_2\ M_1$. The cost of the two policies can be compared as follows.

$$F_w(g) - F_w(\hat{g}) = C(M(f, 0))\mathcal{M}(F_1')^{-1} - C(F_1')\mathcal{M}(M(f, 0))^{-1} = \mathcal{M}(F_1')^{-1}\ \mathcal{M}(M(f, 0))^{-1}\ (\bar{\nu}[M(f, 0)] - \bar{\nu}[F_1']) \quad (4.13)$$

From the assumption $\bar{\nu}[F_1''] < \bar{\nu}[M(f, 0)]$ and Lemma 3,

$$\bar{\nu}[M(f, 0)] > \bar{\nu}[F_1''] \ge \bar{\nu}[F_1']. \quad (4.14)$$

Therefore, $\hat{g}$ is better than g. □

From Theorem 3, it is straightforward to observe that if $\tau_{f,1}(1) = N_f$ for all $f = 1, 2, \ldots, N$, then the optimal GT policy is also optimal within the larger class G^L for both problem **S** and **D**.

The switching index stopping time, $\tau_{f,n}(j)$, can be easily computed for problem **D**. Let the candidate stopping time i vary from $\tau_{f,n}(j-1)$ to N_f. If

$$c(f, i)\mu(f, i) \ge \bar{\nu}[(f, 0)\ (f, n) \cdots (f, i-1)], \quad (4.15)$$

then $\tau_{f,n}(j) \ge i$; otherwise, $\tau_{f,n}(j)$ would not achieve the maximum in Equation (4.6). Thus,

$$\tau_{f,n}(j) = \min\{i > \tau_{f,n}(j-1) : \bar{\nu}[(f, 0)\ (f, n) \cdots (f, i)] > c(f, i+1)\mu(f, i+1)\}. \quad (4.16)$$

Using this type of thinking, we can easily deduce the following corollary of Theorem 3:

COROLLARY 2. *For problems* **S** *and* **D** *and any given problem instance, an increase in the mean set-up time for any family f from* D_f *to* D_f' *results in longer candidate stopping times* $\tau'_{(\cdot,\cdot)}(\cdot)$. *The resulting space of potentially optimal policies is equal to or smaller than that for the original problem.*

To justify Corollary 2, we simply observe that in the new problem for any τ we have $\bar{\nu}'[(f, 0)\ (f, n) \cdots (f, \tau)] \leq \bar{\nu}[(f, 0)\ (f, n) \cdots (f, \tau)]$, and thus by Equation (4.6) the result follows. We observe that the scheduling problem is hardest to solve when the set-up times are small (and nonzero) or intermediate in size. When the set-up times decrease to zero, the solution converges to the $c\mu$ rule or SWPT. Similarly, we see from Theorem 3 that as the set-up times go to infinity, a GT policy, which allows each family to set up only once, is optimal.

It is tempting to suppose that the solution to this problem is as simple as determining and sequencing the batches according to nonincreasing switching indices. That is, start with the family with the maximum $\nu_{f,n}(1)$ and continue until the stopping time $\tau_{f,n}(j)$ such that continuation would yield a smaller total reward rate than is available from another job family. The following example contradicts this:

EXAMPLE 1. Consider the following three-family example. Only family h has a set-up time, and it takes one unit.

Job (f, s)	$(h, 0)$	$(h, 1)$	$(h, 2)$	$(k, 0)$	$(k, 1)$	$(\ell, 0)$	$(\ell, 1)\cdots (\ell, 2000)$
$c(f, s)$	0	5	2	0	3	0	0.5
$\mu(f, s)^{-1}$	1	2	2	0	2	0	1

Thus, a naive application of Theorem 1 and Corollary 1 might suggest that a greedy algorithm would yield an optimal job sequence, g, which is $B_{h(1)}B_kB_{h(2)}B_\ell$ where

$$B_{h(1)} = (h, 0)(h, 1) \qquad \bar{\nu}[B_{h(1)}] = 1.66$$

$$B_k = (k, 0)(k, 1) \qquad \bar{\nu}[B_k] = 1.50$$

$$B_{h(2)} = (h, 0)(h, 2) \qquad \bar{\nu}[B_{h(2)}] = 0.66$$

$$B_\ell = (\ell, 1) \cdots (\ell, 2000) \qquad \bar{\nu}[B_\ell] = 0.50$$

Now, consider $\hat{g}$ which is similar to g, but one set-up is eliminated. $\hat{g} = B_{h(1)}(h, 2)B_kB_\ell$. We will show that $\hat{g}$ improves g.

$$F_w(g) - F_w(\hat{g}) = C((h, 2))\mathcal{M}(B_k(h, 0))^{-1} + C(B_\ell)\mathcal{M}((h, 0))^{-1} - C(B_k)\mathcal{M}((h, 2))^{-1}$$

$$= (2)(3) + (2000)(0.50)(1) - (3)(2) = 1000.$$

We conclude that although $\hat{g}$ is superior to g, policy $\tilde{g} = B_k\, B_{h(1)}\, (h, 2)\, B_\ell$ is optimal. $F_w(\hat{g}) - F_w(\tilde{g}) > 0$ because $\bar{\nu}[B_{h(1)}(h, 2)] = 1.40 < \bar{\nu}[B_k] = 1.50$. Since $\bar{\nu}[B_\ell] = 0.50$, which is the lowest, no other policies are optimal as indicated by Theorem 1.

Because the processing time of subsequent jobs is large, switching between job families results in a large increase in the holding cost. In this case, it is better to process all jobs of family h together rather than switching to another family.

4.3. Sufficient Conditions

We proceed to identify further sufficient conditions for either keeping the server with the current family or switching from one family to another. These conditions may be useful in limiting the search required to find an optimal policy and in developing an effective heuristic scheduling policy.

To start, we address the decision of which batch to serve next. Our analysis assumes for convenience that the decision epoch of concern is at time $t = 0$, since the system can always be relabeled (provided that a switch is always required at time 0, the first decision epoch). In Proposition (2), which is proved in Appendix B, we derive a sufficient condition for initiating service of a batch of family f, allowing the stopping time for family f to vary.

PROPOSITION 2. *For either problem* **S** *or* **D** *with* $F \triangleq (f, 0)\ (f, 1) \cdots (f, N_f)$, *if* $\bar{\nu}[F] \geq \nu_{k,1}(1)$ *for every family* $k : k \neq f$, *and* $k \in 1, \ldots, N$, *then there exists an optimal policy that serves a batch of family f first. Moreover, the optimal sequence will begin with the string* $(f, 0)(f, 1) \cdots (f, \tau_{f,1}(1))$.

Notice that the following conjecture which appears similar to Proposition (2) is not necessarily true. With $F \triangleq (f, 0)\ (f, 1) \cdots (f, \tau_{f,n}(j))$, if $\bar{\nu}[F\ (k, 0)] > \nu_{k,1}(1)$ for some j and for every family k: $k \neq f$, and $k \in 1, \ldots, N$,

then one might conjecture that it is optimal to serve batch F next. We have already provided above a counterexample to this hypothesis (see Example 1). In that example, $\bar{\nu}[(h, 0)(h, 1)(k, 0)] = \bar{\nu}[(h, 0)(h, 1)] > \nu_{k,1}(1)$, and $\nu_{\ell,1}(1)$, but it is optimal to serve batch B_k first.

The preceding proposition was stated for the decision at time $t = 0$, but it is useful as a test at any decision epoch. The same is true of the following: Proposition 3 indicates that it is optimal to continue serving family f as long as it provides a per-job reward rate exceeding that available from a batch of another family. Its proof is in Appendix B.

Proposition 3. *Consider either problem* **S** *or* **D** *formulated in § 2, but with one difference. Suppose that at time $t = 0$, family f is already set up. If $c(f, s)\mu(f, s) \geq \nu_{i,1}(1)$ for all $i \neq f$, for $1 \leq s \leq L$, and some $L \in \mathbb{N}$; then it is optimal to consecutively serve jobs $(f, 1) \ldots (f, L)$ (and possibly others) of family f starting at time $t = 0$.*

The example below is presented to show that Proposition 3 is not a necessary condition.

Example 2. Consider sequencing jobs of two families (h and k). Family h has already been set up at time zero, and we will show that it is better to serve the next job of family h although its reward rate is lower than that of the next batch of another family.

Job (i, j)	$(h, 0)$	$(h, 1)$	$(h, 2)$	$(k, 0)$	$(k, 1)$
$c(i, j)$	0	5	4	0	3
$\mu^{-1}(i, j)$	1	1	2	0	1

Jobs $(h, 0)$ and $(k, 0)$ denote the set-up of family h and k. Theorem 1 and Corollary 1 do not contradict the hypothesis that the jobs may be sequenced as $g = (h, 1)(k, 0)(k, 1)(h, 0)(h, 2)$ with $\bar{\nu}[(h, 1)] = 5$, $\bar{\nu}[(k, 0)(k, 1)] = 3$, and $\bar{\nu}[(h, 0)(h, 2)] = 4/3 < \nu_{k,1}(1) = \bar{\nu}[(k, 0)(k, 1)]$. Notice that once job $(h, 1)$ has been served, the reward rate of $(h, 2)$ is lower than that of family k; nevertheless, we show that continuing with family h is optimal. Now, consider another job sequence $\tilde{g}$ which consists of $(h, 1)(h, 2)(k, 0)(k, 1)$. We will show that $\tilde{g}$ improves g.

$$\begin{aligned} F_w(g) - F_w(\tilde{g}) &= C((h, 2))\mathcal{M}((k, 0)(k, 1)(h, 0))^{-1} \\ &\quad - C((k, 0)(k, 1))\mathcal{M}((h, 2))^{-1} \\ &= (4)(2) - (3)(2) = 2. \end{aligned}$$

Equation (4.16) and Theorem 3 justify the following additional sufficient condition, which was previously proved as Property 3(ii) of Webster and Baker (1994).

Proposition 4. *(Property 3(ii) of Webster and Baker) For problem* **D**, *there exists a GT policy that is optimal if the families can be numbered such that $\bar{\nu}[(f, 0)(f, 1) \ldots (f,N_f)] \leq \bar{\nu}[(f, N_f)]$ for $f = 1, \ldots, N$.*

Webster and Baker proceed to state that "there is no need to split [a batch] if jobs are already grouped into families when arranged in SWPT order." In other words, if the families can be numbered such that

$$\bar{\nu}[(f, N_f)] \geq \bar{\nu}[(f + 1, 1)] \quad \text{for } f = 1, \ldots, N - 1, \tag{4.17}$$

then there exists an optimal GT solution. However, this result, stated as Property 3(i) of [21], is incorrect. We provide the following counterexample.

Example 3. Consider two families, h and k, with the following set-up times and holding costs. We will show that even though Condition (4.17) holds, the GT solution is not optimal.

Job(i, j)	$(h, 0)$	$(h, 1)$	$(h, 2)$	$(k, 0)$	$(k, 1)$
$c(i, j)$	0	1	0.1	0	4
$\mu^{-1}(i, j)$	1	1	1	9	1

The reward rate of the only job of family k is higher than the reward rate of the first job of family h, i.e., $c(k, N_k = 1)\mu(k, N_k = 1)^{-1} = 4 \geq c(h, 1)\mu(h, 1)^{-1} = 1$. Webster and Baker's Proposition 3(i) asserts that there exists a GT policy that is optimal, which we will denote as g^{GT}. However, this is not the case, because policy g^* is optimal. The job sequence and the cost of both policies are as follows.

$$g^{GT} = (k, 0)(k, 1)(h, 0)(h, 1)(h, 2),\ F_w = 53.3$$

$$g^* = (h, 0)(h, 1)(k, 0)(k, 1)(h, 0)(h, 2),\ F_w = 51.4$$

5. The Makespan-constrained Weighted Flowtime Problem

In practical applications, we are concerned not only with minimizing the total holding cost (F_w problem), but also with constraining the total processing and set-up time (makespan, denoted as T). In this section, we

focus our attention exclusively on the deterministic setting. This allows us to avoid the complexities that enter when a makespan target cannot be met with certainty and when optimal dynamic policies become sensitive to the remaining time. Although the F_w problem with a makespan constraint is generally very difficult to solve to optimality, we have been able to establish an important result. To the best of our knowledge, this result has not been identified before. We state the result in the following theorem:

THEOREM 4. All the properties and sufficient conditions stated in § 4 hold with or without the makespan constraint.

To see why this result is true, consider the following. For a feasible solution to exist, the prespecified makespan, T, must be at least as large as the makespan of a GT policy. Therefore, the constraint only restricts the number of set-ups in addition to the GT policy and, thus, the number of batches for each family. None of the structural properties of § 4 (either for **S** or **D**) specify the number of batches each family may have. Moreover, in the proofs of these properties, the modified policy, typically denoted as $\tilde{g}$, $\hat{g}$, g^{ℓ}, or g^{r}, is always constructed in such a way that it never introduces an additional set-up. As a result, the proofs as given justify the results, even with a makespan constraint.

While all these properties provide useful characterizations for optimal policies, Theorem 3 is especially helpful in constructing an optimal policy because it limits our search to batches that stop at the candidate stopping times. The algorithm below is an example of how the stopping time concept can be used.

We begin with either problem **S** or **D** with 2 families. Assume the jobs sharing a common job vector (f, s) have been grouped into a single job according to Proposition 1. Let slack (s) be defined as $s = T - \Sigma_f\{D_f + M((f, 1)(f, 2) \ldots (f, N_f))^{-1}\}$. Suppose the slack is of such a size that one and only one additional set-up can be added to the GT solution for any family. Let the two families be labeled h and k such that the reward rate of family h is greater than or equal to family k, i.e., $\bar{\nu}[(h, 0)\,(h, 1) \ldots (h, N_h)] \geq \bar{\nu}[(k, 0)(k, 1) \ldots (k, N_k)]$. With only one split or less, there are three possible batch configurations: B_hB_k, $B_{h(1)}B_kB_{h(2)}$, and $B_{k(1)}B_hB_{k(2)}$, where $B_{h(i)}$ $(B_{k(i)})$ is the batch of family h (k). With the use of Theorem 3, the following algorithm specifies where to split the family if the splitting can potentially improve the policy.

ALGORITHM. (Two families and one additional set-up permitted; $\bar{\nu}[(h, 0)(h, 1) \ldots (h, N_h)] \geq \bar{\nu}[(k, 0)(k, 1) \ldots (k, N_k)]$.)

1. Compute an optimal policy of the form $B_{h(1)}B_kB_{h(2)}$.
   - 1.1 Set up for family h and process the jobs until the first stopping time, $\tau_{h,1}(1)$ (as defined in Equation 4.6).
   - 1.2 At every stopping time, $\tau_{h,1}(j)$ for $j \geq 1$, test if continuing with family h is better than switching to the other family as follows: Let $\hat{n} \triangleq \tau_{h,1}(j) + 1$. If continuing to the next stopping time, $\tau_{h,1}(j + 1)$, gives a higher reward rate than $\bar{\nu}[B_k(h, 0)]$, i.e., $\bar{\nu}[(h, \hat{n}) \ldots (h, \tau_{h,1}(j + 1))] \geq \bar{\nu}[B_k(h, 0)]$, then process job $(h, \hat{n})$ through $\tau_{h,1}(j + 1)$. Otherwise, switch to family k. Serve all of family k, and then switch back to family h to complete it. Repeat this step until switching to family k, or all jobs of family h have been processed. If the splitting of family h does not improve the policy, this step will yield a GT policy.
   - 1.3 Compute the improvement of this policy over the GT solution:

$$\begin{aligned}\Delta F_w^h &\triangleq F_w(B_hB_k) - F_w(B_{h(1)}B_kB_{h(2)})\\ &= \{\bar{\nu}[B_k] - \bar{\nu}[B_{h(2)}]\}\mathcal{M}(B_k)^{-1}\mathcal{M}(B_{h(2)})^{-1}\\ &\quad - D_hC(B_{h(2)}) - D_hC(B_k)\end{aligned}$$

2. If $\bar{\nu}_{k,1}(1) \leq \bar{\nu}[B_h]$, either the GT solution or $B_{h(1)}B_kB_{h(2)}$ is optimal. Otherwise, search for the best policy of the form $B_{k(1)}B_hB_{k(2)}$ by repeating Steps 1.1 to 1.2 with the role of h and k reversed, and compute

$$\begin{aligned}\Delta F_w^k &\triangleq F_w(B_hB_k) - F_w(B_{k(1)}B_hB_{k(2)})\\ &= \{\bar{\nu}[B_{k(1)}] - \bar{\nu}[B_h]\}\mathcal{M}(B_{k(1)})^{-1}\,\mathcal{M}(B_h)^{-1}\\ &\quad - D_kC(B_{k(2)}).\end{aligned}$$

3. Select an optimal policy with one split or less

by comparing ΔF_w^h and ΔF_w^k. If both terms are nonpositive, then the GT solution is optimal.

We only need to prove that Step 1.2 provides a necessary and sufficient condition for determining the batches of family h. The proof is done by a simple interchange argument and shown in Appendix B. In Step 2, $\bar{\nu}_{k,1}(1) \leq \bar{\nu}[B_h]$ implies, by Lemma 2, that no substring of B_k can achieve a reward rate greater than B_h. Thus, $B_{k(1)}B_hB_{k(2)}$ cannot be optimal. The remainder of the algorithm follows by direct computation.

With the use of the stopping time concept and Theorem 3, the algorithm above can be extended to N families with at most one additional set-up complemented to the GT solution. However, as we saw above, an exhaustive search is required. The calculations are straightforward and some sufficient conditions can be developed to reduce the computational effort.

The F_w problem with the makespan constraint becomes very complicated when any number of set-ups is allowed. With the makespan constraint, a policy with a greater number of set-ups may be better or worse than a policy with a smaller number of set-ups. Moreover, a policy that starts with a batch, h, of high value of $\bar{\nu}_{h,1}(1)$, may be worse than a policy that begins with the batch, say k, of lower value for $\bar{\tau}_{k,1}(1)$ as shown in Example 4. An exhaustive search of all possible candidate stopping times (or a branch and bound scheme) may be unavoidable to find an optimal policy. For practical purposes, a heuristic approach appears more appropriate than an exhaustive search.

Example 4. Consider sequencing jobs of two families (h and k). We will show that $F_w(B_{k(1)}B_hB_{k(2)}) < F_w(B_{h(1)}B_{k(1)}B_{h(2)}B_{k(2)}) < F_w(B_hB_k) < F_w(B_{h(1)}B_kB_{h(2)})$. The families are defined in order as

$$\begin{aligned} g_{GT} &= (h,0)(h,1)(h,2)(h,3)(h,4)(k,0)(k,1)\\ &\quad (k,2)(k,3),\\ g_1' &= (h,0)(h,1)(h,2)(k,0)(k,1)(k,2)(k,3)\\ &\quad (h,0)(h,3)(h,4),\\ g_2 &= (h,0)(h,1)(h,2)(k,0)(k,1)(k,2)(h,0)(h,3)\\ &\quad (h,4)(k,0)(k,3)\\ g_1'' &= (k,0)(k,1)(k,2)(h,0)(h,1)(h,2)(h,3)(h,4)\\ &\quad (k,0)(k,3) \end{aligned}$$

Job(i,j)	$(h,0)$	$(h,1)$	$(h,2)$	$(h,3)$	$(h,4)$	$(k,0)$	$(k,1)$	$(k,2)$	$(k,3)$
$c(i,j)$	0	6	5	2	1	0	4	3	1
$\mu^{-1}(i,j)$	1	1	1	1	1	0	1	1	6

The GT solution is g_{GT} with F_w of 98. One extra set-up of family h (for which $\bar{\nu}_{h,1}(1) = 3.66$ while $\bar{\nu}_{k,1}(1) = 3.5$) increases the cost to 109, resulting in policy g_1'. However, one extra set-up of family k yields a better policy, g_2 with F_w of 94. From this example, we see that there is no simple test to determine which family should be split, if only one extra set-up is allowed. Nevertheless, the optimal policy is g_1'' in which family k is split into two batches and family h is set up once, with F_w of 91.

6. Conclusions

We offer the following managerial insights, which illustrate principles extracted from our technical analysis. The primary insights surround the question of when it is reasonable to apply a very simple policy with few set-ups per family. In the extreme case, this would be the optimal Group Technology (GT) policy. We have learned that the jobs of a given family tend to be broken up as a *small* number of batches to form an optimal schedule when the following conditions apply (as opposed to a larger number of batches when the following conditions do not apply).

1. Optimal policies will tend toward few batches per family when there is little variation in inventory-holding costs and mean processing times—particularly when the costs and processing times are paired so that the product $c_i\mu_i$ (a reward rate index of holding cost times service rate) varies little in i.
2. A few batches per family will tend to be optimal when the set-up times are large and hence the resulting penalty for a set-up is large.
3. There will tend to be fewer batches per family when constraints are placed on the order in which jobs within a family may be scheduled. Specifically, the inclusion of a precedence ordering across jobs within a family (problem **S**) tends to limit the amount of switching. This is seen quite clearly in the reward rate measure of Equation

(4.4)—Equation (4.6). In problem **D** the ordering of jobs ranges from that with the maximum $c\mu$ index to the job with the minimum $c\mu$ index. With precedence constraints, problem **S**, there is only one possible ordering of precedence constraints within family f with the same ordering as problem **D**. On the other hand, there are $N_f!$ possible precedence constraint orderings, each of which has an equal or lesser difference in reward rate across successive batches (a direct consequence of the reward rate of a batch being a weighted average of the individual reward rates of the jobs it contains).

4. An optimal policy tends to have fewer batches when there is a relatively short makespan constraint. This is a simple consequence of the fact that the makespan of a policy increases linearly with the number of set-ups (equivalently, batches).
5. As the number of job families decreases, there will tend to be fewer batches per family. In order to justify the optimality of scheduling family f with two or more batches, the first two of which have reward rates $\bar{\nu}(B_{f,1})$ and $\bar{\nu}(B_{f,2})$ respectively with $\bar{\nu}(B_{f,1}) > \bar{\nu}(B_{f,2})$, there must be at least one other family h with a batch i of h with reward rate $\bar{\nu}(B_{h,i})$ such that $\bar{\nu}(B_{f,1}) > \bar{\nu}(B_{h,i}) > \bar{\nu}(B_{f,2})$. The likelihood of such an event increases as the number of families increases.

In addition, we have generated technical insights and results of value to researchers and practitioners interested in using our results to generate schedules or scheduling heuristics and implementing an optimization procedure. Set-up times introduce a difficult issue into the scheduling problem and require the introduction of a switching index in addition to the standard Gittins index for one to identify the candidate stopping times at which a batch may be ended under an optimal policy. Both the Gittins indices and the switching indices are highly intuitive measures of reward rate and are calculated in terms of holding cost per unit time. Gittins indices do not include the set-up time, while switching indices do. We learned from them that the reward rates, which are equally sensitive to both holding costs and service rates, are key to determining the relative ordering of two fixed batches. However, the set-up times introduce an additional cost equal to the product of the mean set-up time and the total holding cost of all jobs uncompleted at the time of the switch. This greatly complicates the issue of whether to split a family, because the effect of an additional set-up time depends upon all the other jobs present in the system at the time the switch is made. Simple index policies are no longer optimal; rather, all the families are interrelated through the set-ups. The switching indices do, however, limit the number of switches one needs to consider. They elegantly show how the set of potentially optimal policies decreases monotonically as the mean set-up time of a family increases. We have offered several examples that illustrate the subtleties of the problem and suggest approaches for future heuristics. The reward rate approach also makes it easier to understand sufficient conditions for not splitting a family into batches and conditions under which a group technology policy is optimal. All the properties and insights obtained under the expected weighted flowtime criterion carry over directly to the deterministic case with a makespan constraint. Computationally, the space of feasible solutions is reduced, making the problem easier. We demonstrated the use of the switching index in an algorithm for the special case of the makespan-constrained weighted flowtime problem with two families.

We hope that future work will use these new structural perspectives in a heuristic policy. In particular, the results for the case with a makespan constraint are suited for heuristic implementation in a system with job arrivals over time. Concerning our insights toward an effective heuristic approach to scheduling, we offer a few observations. First, we have found that it is both simple and helpful to construct the optimal GT policy. Based upon the amount of slack time available for additional set-ups, more complex alternatives can be constructed. One approach is to add extra set-ups one at a time to build up a heuristic policy. This is similar to the process analyzed in detail in §5. The stopping time indices should be used to limit the search over possible break-points for the batches. We expect that the savings gained by splitting a family into two batches increases when (1) the set-up time for that family is

small; (2) the reward rate for the first batch of the family is significantly larger than the reward rate for the remaining batch of the same family, and also the reward rates for at least some of the other families lies between these two extremes (cf. Theorem 1); and (3) there is a significant holding cost contribution to all three of the following: the first batch of the split family, the second batch of the split family, and the jobs which come between these split batches (as ordered by Theorem 1). When set-up times are very large, it is hard to improve on a GT policy, and a heuristic should work well. If set-up times are very small, the optimal policy will approach the $c\mu$ rule, which can also be generated using the above approach.

From the perspective of flexible manufacturing systems or agile workforces with cross-trained workers, further work is needed to translate our insights into practicable heuristic policies. Where human workers are concerned, simple heuristics will be preferable and the group technology approach or other simple rules of thumb may be especially attractive.[1]

Appendix A. Glossary of Notation

B_f	=	batch of family f (a string with a set-up and job services of family f)
$c(f, s)$	=	holding cost of job (f, s)
$C(M)$	=	sum of all holding costs in substring M
D	=	dynamic problem
D_f	=	set-up time of family f
(f, s)	=	job of family f and item s
$f_g(i)$	=	flowtime of the ith job under g
$F_w(g)$	=	weighted flow time of policy g, or the cost of policy g
g	=	a policy (equivalently a list or string of job vectors)
G^L	=	class of open-loop or list policies
G^{PM}	=	class of pure Markov policies
$\mathcal{M}(M)$	=	effective service rate for all jobs in substring M
$\mu(f, s)^{-1}$	=	mean service time of job (f, s)
$n(f, s)$	=	number of jobs with job vector (f, s)
N	=	number of families
N_f	=	number of jobs in family f
$\vec{r}[(f, s)]$	=	$c(f, s)\,\mu(f, s)$, the reward rate of job (f, s)
$\vec{r}[B_f]$	=	$C(B_f)\mathcal{M}(B_f)^{-1})$, the reward rate of batch of family f, B_f
$\nu_{f,n}(j)$	=	jth largest reward rate of family f starting from job n (includes a set-up)
T	=	pre-specified makespan in the constraint
$\tau_{f,n}(j)$	=	jth candidate stopping time of family f starting from job n (includes a set-up)
s	=	slack in the processing and set-up time
S	=	static problem

[1]This material is based on work supported by the National Science Foundation under grants DMI-9522795 and DMI-9732868. The authors wish to thank the editor, Leroy Schwarz, and two anonymous referees for many helpful comments. They also thank Mark Spearman for a number of excellent ideas that have improved both the content and the exposition of the paper.

Appendix B: Proofs of Selected Results

Reduction of the Stochastic Problem to a Deterministic Problem: Having described the problem of minimizing the total infinite horizon expected cost of a dynamic control policy, we now detail the dynamic programming-based argument that justifies reducing the policy space to the class of list policies. We give the details for problem **D**, while the case of problem **S** is similar. For all times $t \in \mathbb{R}^+$ with $\underline{x} \triangleq [x_{f,s,k}]$ the state space can be defined as $\{(\underline{x}, \gamma)$: $x_{f,s,k} \in \{0, 1\}, f \in \{1, 2, \ldots N\}, s \in \{1, 2, \ldots, N_f\}, k \in \{1, 2, \ldots, n(f, s)\}, \gamma \in \{1, 2, \ldots, N\}\}$. Here at time t, $\underline{x} = [x_{f,s,k}]$ is an array in which $x_{f,s,k}$ equals 1 if the kth job with vector (f, s) (of which there are $n(f, s)$ jobs) is not yet processed at time t; otherwise $x_{f,s,k}$ equals 0. We let γ denote the job family that is set up at t. For this state, admissible control actions are of the form (β, μ_1, μ_2), where $\beta \in \{1, 2, \ldots, N\}, \mu_1 \in \{0, 1, 2, \ldots, N_\beta\}$, and $\mu_2 \in \{1, 2, \ldots, n(\beta, u_1)\}$. Action $(\beta, u_1 = 0, u_2)$ denotes the set-up of family $\beta \in \{1, 2, \ldots, N\}$, and action $(\beta = \gamma, u_1, u_2)$ denotes the service of job (γ, u_1, u_2), which is admissible if $x_{\beta,u_1,u_2} = 1$. Let $\mathcal{A}(\underline{x}, \gamma)$ denote the set of available actions in state $(\underline{x}, \gamma)$. The costs are nonnegative, the state and action spaces are finite (N and $N_{(.)}$ are finite), and discounting is absent. Thus, Proposition 11 in Chapter 5 of Bertsekas (1987) or see Ross (1983) justifies the fact that the following optimality equation is necessary and sufficient for a *stationary, non-randomized* policy to be optimal:

$$V(\underline{x}, \gamma) = \min_{(\beta,u_1,u_2)\in\mathcal{A}(\underline{x},\gamma)} E\Bigg\{\sum_{f=1}^{N}\sum_{s=1}^{N_f}\sum_{k=1}^{n(f,s)} c(f, s)x_{f,s,k}$$

$$[\mathbf{I}\{u_1 \neq 0\}\sigma(\beta, u_1, u_2) + \mathbb{I}\{u_1 = 0\}\delta(\beta)]$$

$$+ V\,\underline{x}\,(\underline{\triangleq} - e_{\beta,u_1,u_2}\,\mathbb{I}\{x_{\beta,u_1,u_2} = 1, u_1 \neq 0\}, \beta)\Bigg\}, \quad (6.18)$$

where $e_{f,s,k}$ is a unit vector with element f, s, k equal to 1, and $V(\underline{x}, \gamma) \triangleq 0$ if $\underline{x} = \underline{0}$. By stationary, we mean that the actions depend only on the state, rather than the time. Nonrandomized means that for each state, one and only one action can be taken. The class of stationary, nonrandomized policies represents a tremendous simplification compared to $\mathcal{G}$, the class of fully dynamic policies. For any $g \in G^{PM}$, observe that the embedded discrete-time queue length process (watched at departures) is a *deterministic* process. By these properties, there exists an optimal policy under which the sequence of states visited is deterministic.

Proof of Lemma 1. Suppose policy g does not exhaustively serve all the jobs sharing the vector (f, s). Viewed as a string, g can be written as $g = M^\circ\,(f, s)\,M\,(f, s)\,M'$, where M contains at least one job with a vector other than (f, s), and may contain jobs of families

other than f. We construct a right-modification of g, $g^r = M^\circ\,(f, s)\,(f, s)\,M\,M'$, as well as a left-modification, $g_\ell = M^\circ\,M\,(f, s)\,(f, s)\,M'$. Both policies are feasible, although $M\,M'$ or $M^\circ\,M$ may contain an unnecessary switch. A graphical depiction of the interchange reveals the following symmetry:

$$F_w(g^r) - F_w(g) = F_w(g) - F_w(g^\ell) = \epsilon \in \mathbb{R}.$$

Thus, if $\epsilon > 0$ ($\epsilon < 0$), then g_ℓ (g^r) strictly improves g. It is possible that $\epsilon = 0$, and that neither g^r nor g_ℓ results in an unnecessary switch. In this case, successively apply either the left or the right interchange until all jobs with vector (f, s) are served consecutively (this argument can be nested to treat any g), and one set-up of family f can be eliminated. The resulting policy costs strictly less than g, since one set-up was eliminated. Thus, an optimal policy must serve each job vector exhaustively. □

Proof of Lemma 3. From the Definition of the stopping time, (4.5) with $F^\circ = (f, 0)(f, n)\,(f, \tau_{f,n}\,(j-1))$,

$$\bar{\nu}[F^\circ F'] \le \bar{\nu}[F^\circ F' F'']$$

That is, the reward rate of continuing to perform jobs until the stopping time is greater than switching to other job families before the stopping time. By Lemma 2, the above equation can be rewritten as

$$c_1\bar{\nu}[F^\circ] + c_2\bar{\nu}[F'] \le d_1\bar{\nu}[F^\circ] + d_2\bar{\nu}[F'] + d_3\bar{\nu}[F'']$$

where $c_1 + c_2 = 1$ and $d_1 + d_2 + d_3 = 1$. Note that $c_1 > d_1 > 0$ and $c_2 > d_2 > 0$ because the reward rate of the string $F^\circ F' F''$ is an average over a longer period than $F^\circ F'$. Furthermore, $\bar{\nu}[F^\circ] > \bar{\nu}[F']$ by Definition (4.5), the definition of reward rate. Thus,

$$(c_1 - d_1)\bar{\nu}[F^\circ] + (c_2 - d_2)\bar{\nu}[F'] \le d_3\bar{\nu}[F'']$$

$$(c_1 + c_2 - d_1 - d_2)\bar{\nu}[F'] \le d_3\bar{\nu}[F'']$$

$$d_3\bar{\nu}[F'] \le d_3\bar{\nu}[F''].$$ □

Proof of Proposition 2. The proof is immediate by Equation (4.7) that, for $j \in \mathbb{N}$, $\nu_{k,1}(j) \le \nu_{k,1}(1) \le \bar{\nu}[F]$, and by Theorem 1. Since it is possible for jobs after $\tau_{f,1}(1)$ to have a very low reward rate (zero, for example), it may be optimal to serve family f in multiple batches. □

Proof of Proposition 3. It suffices to prove this result for $L = 1$, since the case $L > 1$ can be proved by successively applying the following argument L times.

We may write a policy, g, that does not serve $(f, 1)$ at time $t = 0$ in the form $M\,(f, 0)\,(f, 1)\,M'$ where M does not contain jobs from family f (and therefore begins with a set-up). M can be any string and may contain multiple batches. Now, the first batch of M is some string $(m, 0)(m, 1)\cdots(m, \tau_{m,1}(j))$ for some m and j by Theorem 3. Thus, $c(f, 1)\mu(f, 1) \ge \nu_{m,1}(1) \ge \nu_{m,1}(j)$. Furthermore, Theorem 1 implies that the constituent batches of M all have a reward rate of at most $\nu_{m,1}(j)$ because the successive batches occur in the order of non-increasing reward rate. Therefore, $\bar{\nu}[M(f, 0)] < \bar{\nu}[M] \le \nu_{m,1}(j)$ due to the zero reward of set-up. Thus, $\bar{\nu}[M(f, 0)] < c(f, 1)\mu(f, 1)$, and it is easily shown that the new policy $\bar{g} = (f, 1)\,M\,(f, 0)\,M'$ obtained by an interchange of $(f, 1)$ and M performs better than g. □

Proof of Step (1.2) of Algorithm in Section 4. We begin with the first case in which $\bar{\nu}[(h, \hat{n})\cdots(h, \tau_{h,1}(j+1)] < \bar{\nu}[B_k(h, 0)] < \bar{\nu}[(h, \tau_{h,1}(j-1))\cdots(h, \tau_{h,1}(j))]$, where $\tau_{h,1}(0) \stackrel{\triangle}{=} 1$. Suppose a policy, say g, does not switch to family k at $\hat{n}$, but switches at a later point (the case of switching earlier is treated by the second case, given below). By Theorems 3 and 4, we need only consider switching right after a stopping time. It suffices to consider the case where $\hat{n} \stackrel{\triangle}{=} \tau_{h,1}(j) + 1$, and policy g stops family h at $\ell \stackrel{\triangle}{=} \tau_{h,1}(j+\delta)$ for some j and $\delta \in \mathbb{N}$. It follows from Equation (4.7) that $\bar{\nu}[B_k(h, 0)] > \bar{\nu}[(h, \hat{n})\cdots(h, \ell)]$. Policy g can be improved by postponing jobs $(h, \hat{n})\cdots(h, \ell)$ after B_k, and the new policy is denoted as $\tilde{g}$. The detailed job sequence is as follows:

$$g = (h, 0)(h, 1)\cdots(h, \hat{n})\cdots(h, \ell)B_k(h, 0)(h, \ell+1)\cdots(h, N_h)$$

$$\tilde{g} = (h, 0)(h, 1)\cdots(h, \tau_{h,1}(j))B_k(h, 0)(h, \hat{n})\cdots(h, N_h)$$

It can be shown that

$$\begin{aligned} F_w(g) - F_w(\tilde{g}) &= C(B_k)\mathcal{M}((h, \hat{n})\cdots(h, \ell))^{-1} - C((h, \hat{n}) \\ &\quad \cdots(h, \ell))\{\mathcal{M}[B_k]^{-1} + D_h\} \\ &= C(B_k(h, 0))\mathcal{M}((h, \hat{n})\cdots(h, \ell))^{-1} - C((h, \hat{n}) \\ &\quad \cdots(h, \ell))\mathcal{M}[B_k(h, 0)]^{-1} \\ &= \mathcal{M}((h, \hat{n})\cdots(h, \ell))^{-1}\,\mathcal{M}[B_k(h, 0)]^{-1}\,\{\bar{\nu}[B_k(h, 0)] \\ &\quad - \bar{\nu}[(h, \hat{n})\cdots(h, \ell)]\} > 0. \end{aligned}$$

Thus, policy $\tilde{g}$ is better than g. Note that the above proof works for $\ell = N_h$ as well, since $\bar{\nu}[B_k] \ge \bar{\nu}[B_k(h, 0)]$.

We conclude with the second case. If $\bar{\nu}[(h, \hat{n})\cdots(h, \tau_{h,1}(j+1))] \ge \bar{\nu}[B_k(h, 0)]$, and one switches from h to k (as does $\tilde{g}$), then this policy can be improved by serving these jobs prior to switching (that is, policy g with $\ell \stackrel{\triangle}{=} \tau_{h,1}(j+1)$). The above calculations verify the result. □

Bibliography

Ahn, B., J. Hyun. 1990. Single facility multi-class job scheduling; *Computers Opns. Res.* **17**(3), 265–272.

Asawa M., D. Teneketzis. 1996. Multi-armed Bandits with Switching Penalties. *IEEE Trans. Automatic Control* **41**(3), 328–348.

Baker, K. R. 1993. *Elements of Sequencing and Scheduling*. Dartmouth College, Hanover, NH.

Bertsekas, D. P. 1987. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ.

Bruno, J., R. Sethi. 1987. Task sequencing in a batch environment with setup times. *Found. Control Engin.* **3**, 105–117.

Duenyas, I., M. P. Van Oyen. 1996. Heuristic scheduling of parallel heterogeneous queues with set-ups. *Management Sci.* 42:6, 814–829.

Gittins, J. C. 1989. *Multi-armed Bandit Allocation Indices*. Wiley, New York.

Ham, I., K. Hitomi, T. Yoshida. 1985. *Group Technology: Application to Production Management*. Kluwer-Nijhoff Publishing, Boston.

Ishikida, T., P. Varaiya. 1994. Multi-armed bandit problem revisited. *J. Opt. Theory Appl.* **83** 113–154.

Mason, A. J., E. J. Anderson. 1991. Minimizing flow time on a single machine with job classes and setup times. *Naval Res. Logist.* **38** 333–350.

Monma, C. L., C. N. Potts. 1989. On the complexity of scheduling with batch setup times. *Oper. Res.* **37** 798–804.

Morton, T. E., D. W. Pentico. 1993. *Heuristic Scheduling Systems*. Wiley, New York.

Potts, C. N., L. W. Van Wassenhove. 1992. Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. *J. Oper. Res. Soc.* **43** 395–406.

Ross, S. M. 1983. *Introduction to Stochastic Dynamic Programming*. Academic Press, New York.

Santos, C., M. Magazine. 1985. Batching in single operation manufacturing system. *Oper. Res. Letters* 4:3, 99–103.

Van Oyen, M. P., I. Duenyas, C.-Y. Tsai. 1998. Stochastic sequencing with job families, set-up times, and due dates. *Int'l. J. Systems Sci.*

Van Oyen, M. P., D. G. Pandelis, D. Teneketzis. 1992. Optimality of index policies for stochastic scheduling with switching penalties. *J. Appl. Probab.* **29** 957–966.

Van Oyen, M. P., E. Senturk-Gel, W. J. Hopp. 1997. Performance opportunity for flexible workers. *Proc. Thirty-Fifth Annual Allerton Conference on Comm., Control, and Computing* 10 pages.

Van Oyen, M. P., D. Teneketzis. 1994. Optimal stochastic scheduling of forest networks with switching penalties. *Adv. Appl. Probab.* **26** 474–497.

Varaiya, P., J. Walrand, C. Buyukkoc. 1985. Extensions of the multiarmed bandit problem. *IEEE Trans. Automatic Control* AC-30, 426–439.

Webster, S., K. R. Baker. 1994. Scheduling groups of jobs on a single machine. *Oper. Res.* **43** 692–703.

The consulting Senior Editor for this manuscript was Mark Spearman. This manuscript was received on and was with the authors 73 days for 2 revisions. The average review cycle time was 73.6 days.