INFORMS Journal on Computing
Vol. 24, No. 1, Winter 2012, pp. 148–164
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1100.0433

Computing Near-Optimal Policies in Generalized Joint Replenishment

Daniel Adelman
Booth School of Business, University of Chicago, Chicago, Illinois 60637,
dan.adelman@chicagobooth.edu

Diego Klabjan
Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, Illinois 60208,
d-klabjan@northwestern.edu

We provide a practical methodology for solving the generalized joint replenishment (GJR) problem, based on a mathematical programming approach to approximate dynamic programming. We show how to automatically generate a value function approximation basis built upon piecewise-linear ridge functions by developing and exploiting a theoretical connection with the problem of finding optimal cyclic schedules. We provide a variant of the algorithm that is effective in practice, and we exploit the special structure of the GJR problem to provide a coherent, implementable framework.

Key words: approximate dynamic programming; piecewise-linear ridge functions; generalized joint replenishment
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received July 2009; revised July 2010; accepted August 2010. Published online in *Articles in Advance* February 2, 2011.

1. Introduction

In the generalized joint replenishment (GJR) problem, a controller continuously monitors inventories for a finite set of items $\mathcal{I}$. An item may represent a product, a single-item location, or a product–location pair. The inventory of each item $i \in \mathcal{I}$ is infinitely divisible, is consumed at a constant deterministic rate of $0 < \lambda_i < \infty$, and costs the firm $0 \le h_i < \infty$ per unit per time to hold. It also cannot exceed a maximum allowable inventory level of $0 < \overline{X}_i \le \infty$. For each i, to avoid degenerate cases, we assume that either $h_i > 0$ or $\overline{X}_i < \infty$ (or both). As inventories continuously deplete, the controller may at any time replenish a subset $I \subseteq \mathcal{I}$ of items, which incurs an ordering cost of $0 < C_I < \infty$ and is completed instantaneously. Without loss of generality, we assume $C_{I_1} \le C_{I_2}$ if $I_1 \subseteq I_2$, because otherwise the controller can replenish I_1 by executing I_2 without replenishing items $I_2 \setminus I_1$. Although we can accommodate different item sizes, we assume for simplicity that all demands and inventories are measured in the same units, e.g., liters, and that no more than $0 < \bar{A} \le \infty$ total units can be replenished across all items in a single replenishment. The controller's problem is to minimize the long-run time average cost, subject to allowing no stockouts.

To help us motivate and illustrate ideas, we carry the following numerical example throughout the paper.

Example 1. Consider an instance of the deterministic inventory routing problem with two customers, labeled A and B. They are geographically distributed at distances as shown in Figure 1, relative to the storage depot from which all deliveries commence. There are three possible replenishment subsets with costs $C_{\{A\}} = 50$, $C_{\{B\}} = 50$, and $C_{\{AB\}} = 85$. Customers A and B have storage limits of $\overline{X}_A = 2$ and $\overline{X}_B = 3$, respectively, and demand rates of $\lambda_A = \lambda_B = 1$. There are no holding costs. Delivery vehicles have a storage limit of $\bar{A} = 5$, which is effectively infinite because $\overline{X}_A + \overline{X}_B \le \bar{A}$.

We can immediately identify three schedules, although there are uncountably infinite others.

- **Direct shipment:** Whenever a customer stocks out, replenish them with a vehicle that visits only that one customer and fills them up. Under this schedule, customer A will receive replenishments at rate $\lambda_A/\overline{X}_A = 1/2$ with an average cost rate of $50/2$. Likewise, customer B will receive replenishments at rate $\lambda_B/\overline{X}_B = 1/3$ with an average cost rate of $50/3$. Therefore, the total cost rate of this schedule equals $50/2 + 50/3 \approx 41.667$.
- **Universal shipment:** Synchronize replenishments so that both customers A and B always stock out at the same time and are replenished together with the same vehicle. The cheapest such schedule replenishes two units to each customer and generates a cost rate of $(25 + 35 + 25)/2 = 85/2 = 42.5$. Hence, this schedule is less desirable than direct shipment.
- **Four-step cyclic schedule:** Consider the following sequence of replenishments, starting from state

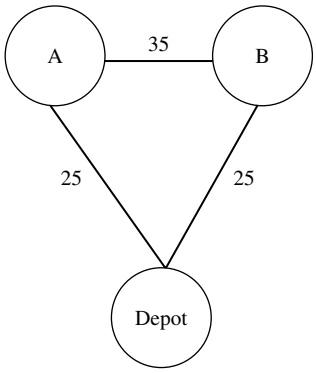

Figure 1 Numerical Example Based on the Deterministic Inventory Routing Problem

vector $x = \langle 0, 0\rangle$ denoting both customers are stocked out. The first replenishment delivers quantity 2 to customer A and quantity 3 to customer B, which we denote by $A2B3$. This instantaneously brings the inventories up to level $\langle 2, 3\rangle$, and then customer A stocks out again after two time units when customer B has inventory level 1; i.e., the next state is $\langle 0, 1\rangle$. Now fill customer A up to his maximum inventory level. Continuing in this manner, we produce the following cyclic schedule having time length 6 and which eventually leads again to state vector $x = \langle 0, 0\rangle$:

$$\langle 0, 0\rangle \xrightarrow{\textbf{A2B3}} \langle 0, 1\rangle \xrightarrow{\textbf{A2}} \langle 1, 0\rangle \xrightarrow{\textbf{B3}} \langle 0, 2\rangle \xrightarrow{\textbf{A2}} \langle 0, 0\rangle.$$

The cost rate of this cyclic schedule is $(85 + 50 + 50 + 50)/6 \approx 39.167$. This beats the other two policies by 6%.

Is the above four-step cyclic schedule optimal? Until now, there has not existed a general, practical method for either constructing this schedule, or even proving that it is optimal. That is the main purpose of this paper.

GJR can be modeled as an infinite-dimensional linear program. Klabjan and Adelman (2007) devised a theoretical solution algorithm and proved that it converges. This algorithm is based on making a value function approximation using piecewise-linear ridge functions, which are shown to be dense in the space of all Borel measurable bounded functions, except on a set of measure zero, and are therefore able to approximate nearly any function arbitrarily closely. Such functions are superpositions of linear functionals and piecewise-linear functions. The algorithm solves the problem by automatically generating new basis functions as part of the overall value function approximation. This is the first and only known algorithm in approximate dynamic programming to do this, and it is the first to even provide a theoretical viewpoint to dynamically generate basis functions. Rather than blindly refining a discrete lattice, or mesh, for state-action spaces to approximate the dynamic programming problem, we show how to judiciously add breakpoints to our functional approximation using duality theory. However, the algorithm's convergence is based on solving a hard nonlinear integer programming problem, termed in this document as the (r, b)-*generation problem*, to optimality; in fact, there is no practical way of solving this problem optimally. Furthermore, even if this problem could be solved to optimality, we lack an analysis of the convergence rate in terms of number of the breakpoints and ridge vectors needed to achieve a given optimality tolerance.

In the present paper, we provide a practical approximate algorithm that generates a feasible solution to the (r, b)-generation problem. We prove that it is guaranteed to improve the value function approximation by cutting off the current solution to the semi-infinite linear program that arises under the ridge function approximation, thereby improving the resulting lower bound. We do this by providing new theory that connects the underlying optimization problems to the problem of finding an optimal cyclic schedule. Based on extensive numerical experience, we then modify this basic algorithm to one that works well in practice for GJR. To our knowledge, this is the first practical algorithm for constructing provably optimal schedules for GJR.

In principle, our ideas can be applied to any deterministic semi-Markov decision process (SMDP) on continuous spaces. We present many ideas in this generality and discuss their broader applicability in the conclusion section. In the special case of the GJR problem, we provide mixed-integer programming formulations of the control policy and separation problems. We pull the various components together to formulate a single algorithmic approach for computing near-optimal control policies. We demonstrate numerical performance on several problem instances of the GJR problem and show superiority relative to an optimal fixed-partition policy as well as near optimality relative to the best lower bound obtained. The fixed partitioning policy a priori partitions the customers into clusters and replenishes all customers in a cluster with a single dispatch.

We provide the first practical algorithm for the generalized joint replenishment problem that produces solutions that are either provably optimal or have a performance guarantee below 2%. Another contribution of our work is the generality of the studied problem and algorithm. Most of the prior work either assumes holding cost or storage/replenishment constraints, but not both. In addition, much of the past work assumes so-called major/minor fixed costs. We can handle arbitrary costs not requiring standard properties such as submodularity or the major/minor structure on subsets of items to replenish. In designing the algorithm, we also provide a few results pertaining to cyclic schedules that repeat states in a

cyclical form, which build an intuitive understanding of how to cut off noncyclic solutions (solutions that never repeat a state) through additional breakpoints in piecewise-linear functions. These results are interesting in their own right.

To make this paper as self-contained as possible, we review in §2 the basic models and value function approximation taken from previous work. In §3 we present specialized models of the separation and policy subproblems for GJR and specify the algorithm for policy simulation. In §4 we make theoretical connections between the problem of finding (near) optimal cyclic schedules and (r, b)-generation. We use this theory to help motivate a practical algorithm, presented in §5. Last, we present numerical results in §6. We conclude the introduction with a literature review.

1.1. Literature Review

Theoretical and algorithmic approaches to deterministic inventory control problems, based on a mathematical programming approach to approximate dynamic programming, originated in Adelman (2003). Adelman considers the special case of the deterministic inventory routing problem without holding costs (see Dror 2005 for a recent review of the inventory routing literature). It makes several novel contributions that include formulating the problem as a semi-Markov decision process and studying the theoretical structure and algorithmic performance that arise under an affine value function approximation. Although policy performance was shown to be good in numerical instances, in some instances there still remained a significant gap between the policy performance and lower bound. The work lacks a methodology for closing this gap, and it also does not model holding costs. We provide such a methodology in this paper, and our numerical results demonstrate that usually it is the affine lower bound that is weak rather than the affine policy. This is important because the affine value function approximation case is numerically efficient and scalable.

Other work on approximate dynamic programming includes Powell (2007), who considers a simulation-based approach for updating value function approximations using policy gradients. Schweitzer and Seidmann (1985) were the first to consider the linear programming approach to computing functional approximations to the dynamic programming value function. This was more recently considered in de Farias and Van Roy (2003).

Until recently, no one had even proven the existence of an optimal policy for GJR. Adelman and Klabjan (2005) formulate the problem with holding costs and use new theory developed in Klabjan and Adelman (2006) to prove existence. This new theory establishes the existence of a solution (almost everywhere) to the dynamic programming optimality equation using infinite-dimensional linear programming. This was needed to overcome technical problems that arose as a result of having a deterministic transition kernel and continuous state and action spaces. Adelman and Klabjan (2005) also characterize the relationship between these infinite-dimensional linear programs and cyclic schedules. They prove that cyclic schedules are ϵ-optimal although not necessarily optimal.

2. Preliminaries

2.1. Basic Models

In this section we present the general dynamic programming formulation and associated infinite-dimensional linear programs, which appear in previous works.

2.1.1. Semi-Markov Decision Process. Consider a deterministic SMDP defined on a state space X and action space A, both assumed to be Borel spaces. For each $x \in X$, let $A(x) \subseteq A$ be a nonempty Borel subset that specifies the set of admissible actions from state x. We denote the collection of state-action pairs as $K = \{(x, a)\colon x \in X, a \in A(x)\}$, assumed to be a Borel subset of $X \times A$. Upon taking action a in state x, a cost $c(x, a)$ is incurred, and then the system transitions to some state $s(x, a)$ after a time duration of length $\tau(x, a)$, all with probability one. We assume that $c\colon K \to \mathbb{R}$, $s\colon K \to X$, and $\tau\colon K \to [0, \infty)$ are measurable on K. Let $\{x_n, a_n, t_n\}_{n=0,1,\ldots} \in (K \times [0, \infty))^\infty$ denote any infinite sequence of state-action pairs and transition times. Suppose $f\colon X \to A$ is a measurable decision function that specifies for every $x \in X$ some action $a \in A(x)$. Define the long-run average cost of the system under control f, starting from an initial state $x_0 \in X$, as

$$J(f, x_0) = \limsup_{N\to\infty} \frac{\sum_{n=0}^{N} c(x_n, f(x_n))}{\sum_{n=0}^{N} t_n}.$$

The problem

$$J(x_0) = \inf_{f\colon X\to A} J(f, x_0) \tag{1}$$

finds an optimal decision rule f^* from starting state x_0. Adelman and Klabjan (2005) provide conditions satisfied for the generalized joint replenishment problem and that we assume hold throughout this paper, under which there exists an f^* such that $\rho = J(f^*, x_0) = J(x_0)$ for every $x_0 \in X$. Such a decision rule is said to be *long-run time average optimal* in the class of stationary deterministic decision rules from every starting state.

The generalized joint replenishment problem can be formulated within this framework as a special case. Define the state space as the Borel space

$$X = \{x \in \mathbb{R}_+^{|\mathcal{J}|}\colon \text{ there exists } j \in \mathcal{J} \text{ with } x_j = 0, x \le \overline{X}\},$$

where $\mathbb{R}_+^{|\mathcal{I}|}$ is the nonnegative orthant, and given a finite set S, we denote by $|S|$ its cardinality. It is easy to see that we replenish only if at least one item stocks out. (If all items have positive inventory, we can postpone the replenishment to a later time, and this would not increase the overall cost.) For every state $x \in X$, the action space is the nonempty Borel subset of $A = \{a \in \mathbb{R}_+^{|\mathcal{I}|}\colon \sum_{i\in\mathcal{I}} a_i \le \bar{A}\}$ defined by

$$A(x) = \left\{a \in \mathbb{R}_+^{|\mathcal{I}|}\colon \sum_{i\in\mathcal{I}} a_i \le \bar{A},\ x + a \le \overline{X}\right\}.$$

For all $(x, a) \in K$, the cost of taking action a in state x is $c(x, a)$ given by the sum of fixed ordering costs and holding costs; i.e.,

$$c(x, a) = C_{\text{supp}(a)} + \sum_{i\in\mathcal{I}} \frac{h_i}{2\lambda_i}(2a_i x_i + a_i^2), \tag{2}$$

where we denote by $\text{supp}(a)$ the support set of a. The first term in the holding cost expression accounts for cost accrued by a replenishment of quantity a_i while the existing inventory x_i of item i depletes. For every $(x, a) \in K$, define the transition time by

$$\tau(x, a) = \min_{i\in\mathcal{I}}\left\{\frac{x_i + a_i}{\lambda_i}\right\}, \tag{3}$$

which may equal 0 if not all stocked-out items are replenished. In addition, in the context of dispatching items to customers, this allows a customer to be served several times in the same time instant; however, from the modeling perspective such a case corresponds to two distinct actions and cost accruals. The next inventory state is then given by the function

$$s(x, a) = x + a - \lambda\tau(x, a).$$

The *average cost optimality equation* is

$$u(x) = \inf_{a\in A(x)} \{c(x, a) - \rho\tau(x, a) + u(s(x, a))\} \quad \text{for every } x \in X, \tag{4}$$

where $\rho \in \mathbb{R}$ and $u \in \mathbb{B}(X)$. The constant ρ defined earlier is the optimal loss, whereas $u(x)$ is the bias function and reflects transient costs starting from state x.

2.1.2. Infinite Linear Programming. Let $q = |\mathcal{I}|$. Given a Borel space Z, we denote by $\mathbb{B}(Z)$ the set of all measurable functions on Z. The set of all measures on Z is denoted by $\mathbb{M}(Z)$. Both of these sets can be equipped with a norm and become Banach spaces. Let also $\mathcal{B}(X)$ be the Borel σ-algebra in X.

The optimality equation (4) can be solved almost everywhere through an infinite-dimensional linear program. The primal problem (P) is

$$\inf \int_K c(x, a)\mu(d(x, a)), \tag{5a}$$

$$\int_K \tau(x, a)\mu(d(x, a)) = 1, \tag{5b}$$

$$\mu((B \times A) \cap K) - \mu(\{(x, a) \in K\colon s(x, a) \in B\}) = 0 \quad \text{for every } B \in \mathcal{B}(X), \tag{5c}$$

$$\mu \ge 0, \quad \mu \in \mathbb{M}(K); \tag{5d}$$

and the corresponding dual problem (D) reads

$$\sup \rho \quad \tau(x, a)\rho + u(x) - u(s(x, a)) \le c(x, a) \quad \text{for every } (x, a) \in K\rho \in \mathbb{R},\ u \in \mathbb{B}(X). \tag{6}$$

Dual problem (D) results by relaxing (4) into less-than or equal-to inequalities and considering the fact that a value is less than or equal to the infimum over a set if the value is less than or equal to every element in the set. The decision variables of the primal problem are measures. The objective function (5a) captures the objective function in a spirit similar to more traditional finite linear programs. Constraints (5c) can be interpreted as flow conservation. They state that for each set B the measure of all state-action pairs that transition to a state in B must be equal to the measure of the set "itself"—$B \times A$. The remaining constraint (5b) has a normalizing effect.

We denote by $\min(P)$ and $\max(D)$ the optimal values of the primal and dual programs, respectively. Adelman and Klabjan (2005) prove that strong duality holds between these programs.

2.2. Value Function Approximation

Because we cannot solve (P) exactly, we next describe a general class of value function approximations. What makes this class special is that it is dense, meaning that it has sufficient fidelity to approximate any bounded measurable function on a compact domain arbitrarily closely.

2.2.1. Ridge Functions. Piecewise-linear functions are attractive from a computational standpoint. Single-variate functions are also easy to encode. However, how do we efficiently encode piecewise-linear functions on a high-dimensional domain? Suppose we are given a collection of n continuous piecewise-linear functions $f^j\colon \mathbb{R} \to \mathbb{R}$. Let us associate with each function f^j a *ridge vector* $r^j \in \mathbb{R}^q$, so that for any $x \in X$ and $j \in [J] = \{1, 2, \ldots, J\}$ we can evaluate each function f^j at $r^j x$. In a more general context, ridge functions in mathematics are defined as $f(g(x))$, where g is a functional and f a single-variate function.

Let $\theta \in \mathbb{R}$ and $v \in \mathbb{R}^q$. Then, we approximate the bias function u with an affine function of state plus *piecewise-linear ridge functions*; i.e.,

$$u(x) \approx \theta - vx - \sum_{j\in[J]} f^j(r^j x) \quad x \in X,$$

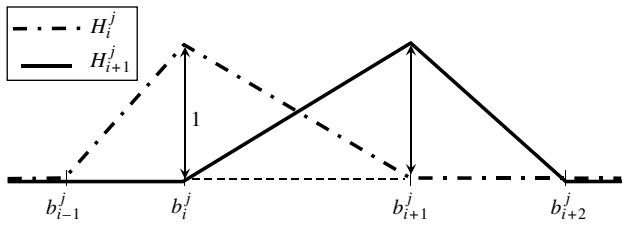

Figure 2 H_i^j **and** H_{i+1}^j **Functions**

where the second term adds piecewise-linear functions, each of which is evaluated at a different scalar product. We subtract terms from the constant θ to facilitate interpretation, because intuitively greater inventory levels should decrease the bias because replenishment costs become less imminent.

We also express the loss as

$$\rho = \sum_{i \in \mathcal{I}} \lambda_i v_i + \hat{\rho}. \tag{7}$$

This produces a decomposition of the loss into a marginal value v_i for each item i, which has managerial meaning. Here, $\hat{\rho}$ has an identical interpretation as ρ except that it refers to the approximate setting, and it is offset by marginal value contributions v.

Rather than work with the functions f^j directly, it is convenient to decompose them into linear combinations of what are called *hat* functions, or *B-splines of degree 1*. Let $b_0^j < b_1^j < \cdots < b_{m_j}^j < b_{m_j+1}^j$ for $j \in [J]$ be real numbers in an interval $[-\Omega, \Omega]$, except that $b_0^j < -\Omega$ and $b_{m_j+1}^j > \Omega$. Each such ordered set of numbers is denoted by b^j, and this is the set of *breakpoints* in function f^j. For each $j \in [J]$ and $i \in [m_j]$, let the hat function $H_i^j \colon [-\Omega, \Omega] \to \mathbb{R}$ be defined as

$$H_i^j(z) = \begin{cases} \dfrac{z - b_{i-1}^j}{b_i^j - b_{i-1}^j} & b_{i-1}^j \le z \le b_i^j, \\ \dfrac{b_{i+1}^j - z}{b_{i+1}^j - b_i^j} & b_i^j \le z \le b_{i+1}^j, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $H_i^j(b_i^j) = 1$ and $\operatorname{supp}(H_i^j) = (b_{i-1}^j, b_{i+1}^j)$; see Figure 2. As the following proposition shows, we can then express

$$f^j(r^j x) = \sum_{i=1}^{m_j} w_i^j H_i^j(r^j x), \quad x \in X,$$

for some set of weights $w_i^j \in \mathbb{R}$, $j \in [J]$, $i \in [m_j]$.

Proposition 1 (Klabjan and Adelman 2007). *The following properties hold for every $j \in [J]$ and $i \in [m_j]$:*

1. *f^j is continuous piecewise linear with breakpoints b_i^j and $f^j(b_i^j) = w_i^j$.*

2. *The slope of f^j in $[b_i^j, b_{i+1}^j]$ is*

$$\frac{w_{i+1}^j - w_i^j}{b_{i+1}^j - b_i^j}.$$

3. *If $z \in [b_i^j, b_{i+1}^j]$, then*

$$\begin{aligned} f^j(z) &= w_i^j H_i^j(z) + w_{i+1}^j H_{i+1}^j(z) \\ &= w_i^j \frac{b_{i+1}^j - z}{b_{i+1}^j - b_i^j} + w_{i+1}^j \frac{z - b_i^j}{b_{i+1}^j - b_i^j}. \end{aligned}$$

Using hat functions for a fixed collection of ridge vectors and breakpoints, our approximation to the bias function becomes

$$u(x) \approx \theta - vx - \sum_{j=1}^{J} \sum_{i=1}^{m_j} w_i^j H_i^j(r^j x) \quad x \in X, \tag{8}$$

where θ, v, and w are unknowns. Without loss of generality, we assume that all ridge vectors are nonzero. Because hat functions encode piecewise-linear functions, we implement them using mixed-integer programming (Croxton et al. 2003). We suppress this in what follows whenever we write f^j. See Klabjan and Adelman (2007) for more discussion on ridge functions.

2.2.2. Semi-Infinite Linear Programs. By plugging approximations (7) and (8) into (D), we obtain the problem of finding weights w and values θ, v that give the largest dual objective value. It is a linear semi-infinite program, which we denote by (DW).

$$\sup \sum_{i \in \mathcal{I}} \lambda_i v_i + \hat{\rho}, \tag{9a}$$

$$\tau(x,a)\hat{\rho} + \sum_{i \in \mathcal{I}} a_i v_i + \sum_{j=1}^{J} \sum_{i=1}^{m_j} w_i^j [H_i^j(r^j s(x,a)) - H_i^j(r^j x)]$$

$$\le c(x,a) \quad \text{for every } (x,a) \in K, \tag{9b}$$

$$\hat{\rho} \in \mathbb{R},\ v \in \mathbb{R}^q,\ w \text{ unrestricted.} \tag{9c}$$

The parameter θ cancels out. The dual (PW) of (DW), which we consider the "primal problem," is

$$\inf \sum_{(x,a) \in T} c(x,a) z_{x,a}, \tag{10a}$$

$$\sum_{(x,a) \in T} \tau(x,a) z_{x,a} = 1, \tag{10b}$$

$$\sum_{(x,a) \in T} a_i z_{x,a} = \lambda_i \quad i \in \mathcal{I}, \tag{10c}$$

$$\sum_{(x,a) \in T} H_i^j(r^j s(x,a)) z_{x,a} - \sum_{(x,a) \in T} H_i^j(r^j x) z_{x,a} = 0$$

$$j \in [J],\ i \in [m_j], \tag{10d}$$

$$z \ge 0,\ \operatorname{supp}(z) = T,\ |T| < \infty. \tag{10e}$$

The decision variable $z_{x,a}$ represents the state-action frequency for pair x, a. Requirement (10b) corresponds with the $\hat{\rho}$ variable, (10c) corresponds with the v variables, and (10d) corresponds with the w variables. Under two additional conditions satisfied by the generalized joint replenishment problem, and which we assume in the remainder of this paper, Klabjan and Adelman (2007) show that strong duality holds between these programs; i.e., (PW) and (DW) are solvable and there is no duality gap. Furthermore, because we are restricting the feasible dual space, we have

$$\min(\text{PW}) = \max(\text{DW}) \leq \max(\text{D}) = \min(\text{P}).$$

Example 2. Consider the pair (PW)–(DW) with an affine value function approximation. This means there are no hat functions, i.e., $J = 0$, so that the approximation becomes

$$u(x) \approx \theta - vx \quad x \in X,$$
$$\rho = \lambda v + \hat{\rho}.$$

In our numerical example, an optimal solution sets $\theta = 0$, $v_A = 25$, $v_B = 11.667$, and $\hat{\rho} = 0$, so that $\rho = 25 + 11.667 = 36.667$ provides a lower bound on the long-run average cost rate of any policy. A corresponding dual optimal solution, e.g., to (PW), is

$$z_{\langle 0,0\rangle,\langle 2,3\rangle} = 1/3 \qquad z_{\langle 0,1\rangle,\langle 2,0\rangle} = 1/6.$$

If a schedule can be constructed that corresponds with this solution (or that has a cost rate of 36.667), then it is optimal. Figure 3 attempts to construct an inventory trajectory that implements this z solution. The positive vertical axis corresponds with the inventory of customer A, whereas the negative axis corresponds with the inventory of customer B. Although both replenishments can be executed, they lead to inventory state $\langle 1, 0\rangle$, for which there is no z variable. Hence, the z solution is not implementable, and we can only say that the cost rate of 36.667 is a lower bound whose tightness is indeterminable.

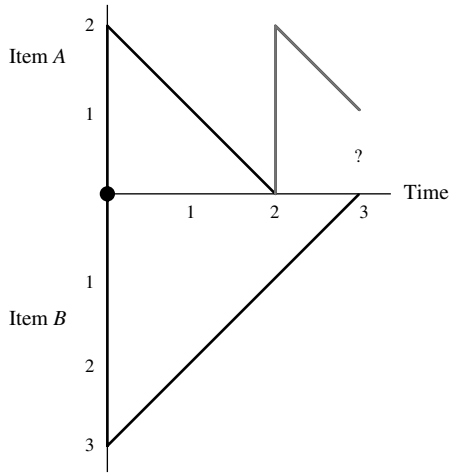

Figure 3 Attempted Trajectory to Implement the z Solution

3. Optimization

Our algorithm relies on solving the primal–dual pair (DW)–(PW). To this end, in this section we assume that the breakpoints and ridge functions are given, and thus we optimize to find the best weights w. These weights uniquely specify an approximate value function. Given weights, we check if the corresponding dual solution obtained by (PW) is feasible to the exact primal problem (P). If it is, the solution is optimal. Otherwise, we use the approximate value function to construct a greedy policy, which looks ahead several decision epochs. If the value of this policy is within a given tolerance to the lower bound, we stop. Otherwise, we study the underlying infeasibility, and based on an infeasible constraint, we either generate a new breakpoint for an existing ridge vector or find a new ridge vector. The procedure is then repeated by again solving (DW)–(PW).

We note that (DW) is solved by row generation because it is a semi-infinite linear (or quadratic in the presence of holding costs) program. We next formulate the basic optimization problems that need to be solved. The first one generates rows for solving (DW), whereas the second one implements the control policy. As we will see, they are closely related.

3.1. Separation Problem

We can solve (DW) using row generation, by solving the separation problem. Given $\hat{\rho}, w$, we want to find the most violated constraint (9b) or assert that none exists. The general separation problem reads as

$$\Phi(\hat{\rho}, w, v) = \min_{(x,a)\in K}\Bigg(c(x,a) - \hat{\rho}\tau(x,a) - \sum_{i\in\mathcal{I}} v_i a_i - \sum_{j=1}^{J}\sum_{i=1}^{m_j} w_i^j [H_i^j(r^j s(x,a)) - H_i^j(r^j x)]\Bigg). \quad (11)$$

When the myopic problem $\min_{(x,a)\in K}\{c(x,a) - \hat{\rho}\tau(x,a)\}$ has a sufficient structure, then the overall separation problem can be formulated as a mixed-integer program because the value function approximation terms are piecewise-linear functions. In the special case of the GJR problem, we can formulate this as the following mixed-integer quadratic program. Let $f^j(r^j x; w^j, b^j)$ denote the value of the jth piecewise-linear function evaluated at the scalar $r^j x$, given weights w^j and breakpoints b^j. The

following model provides a mathematical program for evaluating Φ.

$$\Phi(\hat{\rho}, w, v) = \min_{t, Y, R, a, x, s, U, U'} \sum_{I \subseteq \mathcal{J}} C_I Y_I + \sum_{i \in \mathcal{J}} \frac{h_i}{2\lambda_i}(a_i^2 + 2a_i x_i) - \hat{\rho} t - \sum_{i \in \mathcal{J}} v_i a_i + \sum_{j=1}^{J} [f^j(r^j x; w^j, b^j) - f^j(r^j s; w^j, b^j)]. \quad (12)$$

Subset constraints

$$\sum_{I \subseteq \mathcal{J}} Y_I = 1, \quad (13)$$

$$R_i = \sum_{\{I \subseteq \mathcal{J}: i \in I\}} Y_I \quad i \in \mathcal{J}, \quad (14)$$

$$a_i \le \bar{X}_i R_i \quad i \in \mathcal{J}. \quad (15)$$

State-action constraints

$$s_i = x_i + a_i - \lambda_i t \quad i \in \mathcal{J}, \quad (16)$$

$$x_i + a_i \le \bar{X}_i \quad i \in \mathcal{J}, \quad (17)$$

$$\sum_{i \in \mathcal{J}} a_i \le \bar{A}. \quad (18)$$

Just-in-time constraints

$$x_i \le \bar{X}_i \cdot (1 - U_i) \quad i \in \mathcal{J}, \quad (19)$$

$$\sum_{i \in \mathcal{J}} U_i \ge 1, \quad (20)$$

$$s_i \le \bar{X}_i \cdot (1 - U_i') \quad i \in \mathcal{J}, \quad (21)$$

$$\sum_{i \in \mathcal{J}} U_i' \ge 1, \quad (22)$$

$$U_i \le R_i \quad i \in \mathcal{J}. \quad (23)$$

Miscellaneous

$$Y, R, U, U' \text{ binary}$$

$$x, a, s, t \ge 0.$$

We use the binary variable R_i for convenience: it equals 1 if item i is replenished and 0 otherwise. The binary decision variables Y_I represent the decision of what subset to replenish. With major/minor fixed costs, we have $C_I = a + \sum_{i \in I} b_i$ if $I \neq \varnothing$ and $C_\varnothing = 0$, where a is the fixed cost incurred if any item is replenished, and b_i is an item-specific fixed cost. In this case, we only need the R_is and not Y_Is, which dramatically reduces the size of the subproblem because it avoids enumerating the power set. In this case, we replace (13) with $\sum_{i \in \mathcal{J}} R_i \ge 1$, which requires that at least one item be replenished. We also delete (14).

The first set of constraints (13)–(15) models the selection of a replenishment subset. Constraint (13) requires that we select exactly one subset of items, and (14) ensures that R_i is set properly given the Y_Is. Constraints (15) force the replenishment quantity for item i, denoted by decision variable a_i, to equal 0 if item i is not included in the selected subset. Otherwise, the inequality is implied by (17). If the problem instance has $\bar{X}_i = \infty$ for some i, we can use Proposition 1 in Adelman and Klabjan (2005) to make $\bar{X}_i$ finite without loss of optimality.

The second set of constraints (16)–(18) models feasibility of state-action pairs. The decision variables x_i and s_i represent the inventory of item i at the current and the next decision epoch, respectively. Constraints (16) are standard flow balance constraints, stating that the next inventory state equals the current inventory state plus quantities replenished minus quantities consumed. Constraints (17) and (18) ensure that the upper bounds on inventory states and total replenishment quantity, respectively, are obeyed.

The third set of constraints (19)–(23) ensures that decision epochs are defined by some item stocking out. The binary decision variable U_i equals 1 if item i is stocked out in the initial inventory state, and 0 otherwise. If $U_i = 1$, then constraint (19) ensures that $x_i = 0$. Constraint (20) requires that at least one item be stocked out in the initial inventory state. The pair of constraints (21) and (22) are similar and apply to the next inventory state represented by the s_is. Constraints (23) ensure that at least one of the items replenished is stocked out. Note that (16), (22), and $s \ge 0$ imply that $t = \min_{i \in \mathcal{J}}(x_i + a_i)/\lambda_i$, i.e., the time until the next stockout.

In our implementation, we avoid bringing the nonconvex quadratic term from (2) into the objective function (12) by adding the constraints

$$x_i \le \bar{X}_i \cdot (1 - R_i) \quad i \in \mathcal{J} \quad \text{s.t. } h_i > 0,$$

which model "zero inventory ordering." These constraints force item i to be stocked out when replenished, assuming that the holding cost h_i is strictly positive. Under these constraints, $a_i x_i = 0$, and thus (2), becomes convex. The resulting convex quadratic integer program can be solved using off-the-shelf software, namely, CPLEX. In this case, the optimal objective function value of (DW) obtained provides a lower bound only for policies restricted as such.

3.2. Price-Directed Control Policy

Whereas the optimal objective of (DW) provides a lower bound on the cost of an optimal policy, we can compute an upper bound by simulating the control policy under the resulting ridge function approximation. In particular, given optimal-dual prices $\{\hat{\rho}, w, v\}$, for the current set $\{r, b\}$ of ridge vectors and corresponding breakpoints, the one-step greedy policy is the same as the separation problem, except that the current period's state x is fixed. Algorithm 1 computes an upper bound by simulating the control policy through time, up to at most N decision epochs. If

a state is reached that has been visited before, at some strictly earlier time than now, then the algorithm terminates with a cyclic schedule and an exact upper bound $\mathcal{B}(u)$ based on the value function approximation u. Otherwise, the algorithm returns the average cost obtained over N decision epochs as an approximate upper bound.

Algorithm 1 (Obtaining an upper bound through policy simulation)

1: Choose an initial state x_0.
2: **for** $n=0$ to N **do**
3: In state x_n choose
$a_n^* \in \arg\min_{a\in A(x_n)}\{c(x_n, a) - \rho\tau(x_n, a) + u(s(x_n, a))\}$.
4: Set $t_n = \tau(x_n, a_n^*)$, $x_{n+1} = s(x_n, a_n^*)$.
5: **if** x_{n+1} has been visited before at a step $n' < n+1$
6: Break; cyclic schedule found
7: **end if**
8: **end for**
9: Return $\mathcal{B}(u) = \dfrac{\sum_{m=n'}^{n} c(x_m, a_m^*)}{\sum_{m=n'}^{n} t_m}$, where n' is the first step of the cyclic schedule, or 0 if one not found.

We find empirically that substantial improvements in the policy are achieved by looking ahead multiple decision epochs in Step 3, on a rolling horizon basis. Looking ahead corrects for imperfections in the value function approximation by explicitly considering near-term decisions. Let $\mathcal{N}$ denote the number of periods to look ahead, so that $\mathcal{N}=1$ corresponds with the ordinary one-step greedy policy. Let $x_{n,i}$ be the initial inventory of item i at the start of decision epoch n, which is fixed. In the general setting, the look-ahead control policy chooses a_n^* as the optimal first epoch action obtained by solving

$$\min_{a_n, x_{n+1}, a_{n+1}, \ldots, x_{n+\mathcal{N}}} \sum_{n'=n}^{n+\mathcal{N}-1} (c(x_{n'}, a_{n'}) - \rho\tau(x_{n'}, a_{n'})) + u(x_{n+\mathcal{N}})$$
$$\text{s.t. } x_{n'+1} = s(x_{n'}, a_{n'}), \quad n' \in \{n, \ldots, n+\mathcal{N}-1\}.$$

In the special case of the GJR problem, this becomes (PD):

$$\min_{t, Y, R, a, x, U} \sum_{n'=n}^{n+\mathcal{N}-1} \left(\sum_{I\subseteq\mathcal{I}} C_I Y_{n',I} + \sum_{i\in\mathcal{I}} \frac{h_i\cdot(a_{n',i}^2 + 2a_{n',i}x_{n',i})}{2\lambda_i} - \hat{\rho} t_{n'} - \sum_{i\in\mathcal{I}} v_i a_{n',i} \right)$$
$$- \sum_{j=1}^{J} f^j(r^j x_{n+\mathcal{N}}; w^j, b^j), \tag{24}$$
$$\sum_{I\subseteq\mathcal{I}} Y_{n',I} = 1 \quad n' \in \{n, \ldots, n+\mathcal{N}-1\},$$
$$R_{n',i} = \sum_{\{I\subseteq\mathcal{I}: i\in I\}} Y_{n',I} \quad n' \in \{n, \ldots, n+\mathcal{N}-1\},\ i\in\mathcal{I},$$
$$a_{n',i} \le \bar{X}_i R_{n',i} \quad n' \in \{n, \ldots, n+\mathcal{N}-1\},\ i\in\mathcal{I},$$
$$x_{n'+1,i} = x_{n',i} + a_{n',i} - \lambda_i t_{n'} \quad n' \in \{n, \ldots, n+\mathcal{N}-1\},\ i\in\mathcal{I},$$
$$x_{n',i} + a_{n',i} \le \bar{X}_i \quad n' \in \{n, \ldots, n+\mathcal{N}-1\},\ i\in\mathcal{I},$$
$$\sum_{i\in\mathcal{I}} a_{n',i} \le \bar{A} \quad n' \in \{n, \ldots, n+\mathcal{N}-1\},$$
$$x_{n',i} \le \bar{X}_i\cdot(1-U_{n',i}) \quad n' \in \{n, \ldots, n+\mathcal{N}\},\ i\in\mathcal{I},$$
$$\sum_{i\in\mathcal{I}} U_{n',i} \ge 1 \quad n' \in \{n, \ldots, n+\mathcal{N}\},$$
$$U_{n',i} \le R_{n',i} \quad n' \in \{n, \ldots, n+\mathcal{N}-1\},\ i\in\mathcal{I},$$
$$Y, R, U \text{ binary},$$
$$x, a, t \ge 0,$$
$$x_n \text{ fixed}.$$

Because the initial inventories $x_{n,i}$ are fixed, the nonconvex term for the current period n in the cost function (2) becomes linear in the objective function (24). Because we only implement the first decision a_n, the resulting policy is permitted to replenish items that are not currently stocked out. We can eliminate the nonconvex terms in all other decision epochs by adding the constraints

$$x_{n',i} \le \bar{X}_i\cdot(1-R_{n',i}) \quad n' \in \{n+1, \ldots, n+\mathcal{N}-1\},\ i\in\mathcal{I} \quad \text{s.t. } h_i > 0.$$

This imposes the requirement that items replenished in the future are stocked out. The objective function leaves off the initial value $\sum_{j=1}^{J} f^j(r^j x_n; w^j, b^j)$, and because we unwind the recursion in decision epochs $n, n+1, \ldots, n+\mathcal{N}$, the value function approximation only comes into play for the terminal inventories. Otherwise, the formulation is the multiperiod generalization of the separation problem with x_n fixed and can be solved using CPLEX.

4. Basis Generation

The key to generating a new (r, b) is to exploit the connection between (P) and the problem that optimizes over cyclic schedules.

4.1. Cyclic Schedules

We first provide a general definition of a cyclic schedule.

Definition 1. A sequence $\mathcal{C} = \{(x_n, a_n)\}_{n=0,\ldots,N-1}$ of $N < \infty$ state-action pairs is called a *cyclic schedule* if

$$x_n = \begin{cases} s(x_{N-1}, a_{N-1}) & \text{for } n=0, \\ s(x_{n-1}, a_{n-1}) & \text{for } n=1, \ldots, N-1. \end{cases}$$

The primal problem optimizing over all cyclic schedules (PC) is defined by

$$\inf_{z} \sum_{(x,a)\in T} c(x,a) z_{x,a}, \tag{25a}$$

$$\sum_{(x,a)\in T} \tau(x,a) z_{x,a} = 1, \tag{25b}$$

$$\sum_{a:(x,a)\in T} z_{x,a} - \sum_{\substack{(\bar{x},\bar{a})\in T \\ s(\bar{x},\bar{a})=x}} z_{\bar{x},\bar{a}} = 0 \quad \text{for every } x \in X, \tag{25c}$$

$$z \geq 0,\ \operatorname{supp}(z) = T,\ |T| < \infty. \tag{25d}$$

This model restricts (P) to a measure μ with finite support, denoted here by z. A cyclic schedule $\mathscr{C}$ induces a feasible solution to (PC) by defining $z_{x,a} = 1/\sum_{(x',a')\in\mathscr{C}} \tau(x',a')$ for every $(x,a) \in \mathscr{C}$ and 0 otherwise. We call such feasible solutions to (PC) *cyclic schedule solutions*. See Adelman and Klabjan (2005) for a theory on cyclic schedules. The next proposition states that there are other feasible solutions to (PC) that decompose into multiple cyclic schedule solutions. However, they are dominated by cyclic schedule solutions containing a single cycle.

Proposition 2. *If z is a feasible solution to* (PC), *then there exists a cyclic schedule solution $\bar{z}$ with* $\operatorname{supp}(\bar{z}) \subseteq \operatorname{supp}(z)$ *and with a cost that is not larger.*

Proof. With respect to z, we define the following network N. The nodes of N correspond to states $x \in X$ with the property that there exists an action $a \in A(x)$ with $z_{x,a} > 0$. There is an arc (x,y) from node (state) x to node (state) y if there exists an action $a \in A(x)$ such that $s(x,a) = y$. Because $|\operatorname{supp}(z)| < \infty$, N is a finite network.

Because of (25c), the values of z induce a circulation in N. Every circulation can be decomposed into directed cycles; see, e.g., Ahuja et al. (1993). Let $\mathscr{C}_1, \ldots, \mathscr{C}_k$ be the cycle decomposition of z. By definition, there exist values $K_1, \ldots, K_k$ such that $z_{x,a} = \sum_{\{j\in[k]:(x,a)\in\mathscr{C}_j\}} K_j$ for every $(x,a) \in \mathscr{C}_j$ and for every $j = 1,2,\ldots,k$. Let $\bar{\tau}_j = \sum_{(x,a)\in\mathscr{C}_j} \tau(x,a)$ and $\bar{c}_j = \sum_{(x,a)\in\mathscr{C}_j} c(x,a)$ for $j = 1,2,\ldots,k$. From (25b) we obtain $\sum_{j=1}^{k} \bar{\tau}_j K_j = 1$. The objective value of z is $\sum_{j=1}^{k} K_j \bar{c}_j$.

Consider now the LP:

$$\min \sum_{j=1}^{k} \bar{c}_j t_j$$

$$\sum_{j=1}^{k} \bar{\tau}_j t_j = 1,$$

$$t \geq 0.$$

By the above observations, $t_j = K_j$ for $j = 1,2,\ldots,k$ is a feasible solution to this LP. Let $t_{\bar{j}}^* = 1/\bar{\tau}_{\bar{j}}$ and $t_j^* = 0$ for all $j \neq \bar{j}$ be an optimal basic solution for some index $\bar{j}$. Then, clearly, $c_{\bar{j}} t_{\bar{j}}^* \leq \sum_{j=1}^{k} K_j \bar{c}_j$.

The cyclic schedule solution defined by $\bar{z}_{x,a} = 1/\bar{\tau}_{\bar{j}}$ for every $(x,a) \in \mathscr{C}_{\bar{j}}$, and 0 otherwise, has the desired property. □

By the above proposition, (PC) is the problem of finding the minimum cost cyclic schedule. Let $\inf(PC)$ denote the optimal value of (PC). Note that this value might not be attainable; i.e., (PC), in general, is not solvable. Because cyclic schedules in the GJR problem are ϵ-optimal to (P) and (PC) for every $\epsilon > 0$, we know that $\inf(PC) = \min(P) = \max(D)$; e.g., see Adelman and Klabjan (2005). We call constraints (5c) and (25c) the *flow balance constraints*.

Next, we establish the relationship between (PC) and (PW). We later prove a variant of the converse statement.

Proposition 3. *If z is a cyclic schedule solution to* (PC), *then z is feasible to* (PW) *for any set of ridge vectors and breakpoints.*

Proof. Let z be a cyclic schedule solution to (PC) and denote its support set by T. Because of the flow balance constraints, we have $z_{x,a} = 1/\sum_{(x',a')\in T} \tau(x',a')$ for every $(x,a) \in T$. Note that for every $j \in [n]$, $i \in [m_j]$ and (r^j, b^j), we have

$$\{(x,a) \in T\colon b_i^j \leq r^j s(x,a) \leq b_{i+1}^j\}$$
$$= \{(x,a) \in T\colon b_i^j \leq r^j x \leq b_{i+1}^j\}.$$

These two facts easily show that z satisfies (10d). To check (10c), note that

$$\sum_{(x,a)\in T} a_i z_{x,a} = \frac{1}{\sum_{(x,a)\in T} \tau(x,a)} \sum_{(x,a)\in T} a_i = \lambda_i.$$

This completes the proof. □

This proposition implies that we can solve (PW) by column generation, starting with an initial feasible solution z that corresponds with any cyclic schedule.

Example 3. The universal shipment schedule in Example 1 corresponds with a feasible solution $z_{\langle 0,0\rangle,\langle 2,2\rangle} = 1/2$ to (PW). The four-step cyclic schedule corresponds with the feasible solution to (PW) that sets each of $z_{\langle 0,0\rangle,\langle 2,3\rangle}$, $z_{\langle 0,1\rangle,\langle 2,0\rangle}$, $z_{\langle 1,0\rangle,\langle 0,3\rangle}$, $z_{\langle 0,2\rangle,\langle 2,0\rangle}$ equal to 1/6.

4.2. (r,b) Generation

Suppose z is an optimal solution to (PW) that corresponds with a cyclic schedule. This implies it is an optimal solution to (PC), because it is feasible to (PC) and $\min(PW) \leq \inf(PC)$. Furthermore, because $\inf(PC) = \min(P)$,

$$\mu(\mathscr{K}) = \sum_{\{(x,a)\in\mathscr{K}:z_{x,a}>0\}} z_{x,a} \quad \mathscr{K} \in \mathbb{B}(K)$$

is a feasible optimal solution to (P).

On the other hand, if z does not correspond with a cyclic schedule, then we hope to find a new hat function that, when corresponding constraints are added to (PW), makes the solution z infeasible. Thus we wish to cut off the solution z by restricting the primal feasible space. We call this (r, b)-generation, or basis generation, because we seek a new hat function as part of the basis.

We first define a function that measures the magnitude of the flow imbalance given in (10d) for any hat function. For any $z\colon K \to \mathbb{R}$ with $\text{supp}(z) < \infty$, $r \in \mathbb{R}^q$, and $\bar{b} = (\bar{b}_1, \bar{b}_2, \bar{b}_3)$ with $\bar{b}_1 < \bar{b}_2 < \bar{b}_3$, let

$$g(z, r, \bar{b}) = \Bigg| \sum_{\substack{(x,a)\in\text{supp}(z)\\ \bar{b}_1 \le rx \le \bar{b}_3}} H_{\bar{b}}(rx) z_{x,a} - \sum_{\substack{(x,a)\in\text{supp}(z)\\ \bar{b}_1 \le rs(x,a) \le \bar{b}_3}} H_{\bar{b}}(rs(x,a)) z_{x,a} \Bigg|.$$

Here, we denote by $H_{\bar{b}}$ the hat function with breakpoints $\bar{b}_1, \bar{b}_2, \bar{b}_3$.

It suffices to consider ridge vectors with the infinity norm of 1. Therefore for every $x \in X$ and every ridge vector r with $\|r\|_\infty \le 1$, we have by Cauchy–Schwartz

$$|rx| \le \|r\|_2 \cdot \|x\|_2 \le \sqrt{q} \cdot \text{diam}(X),$$

where $\text{diam}(X) < \infty$ is the diameter of X. Therefore we can select $\Omega = \sqrt{q} \cdot \text{diam}(X)$ (see §2.2.1 for the role of Ω). We denote by $-\Omega \le \bar{b} \le \Omega$ the requirement $-\Omega \le \bar{b}_1 < \bar{b}_2 < \bar{b}_3 \le \Omega$.

In full generality, given a solution z to (PW), we can formulate the (r, b)-generation problem as

$$\max_{\substack{\|r\|_\infty \le 1\\ -\Omega \le \bar{b} \le \Omega}} g(z, r, \bar{b}). \tag{26}$$

The idea is to find a new $(r, \bar{b})$ such that the corresponding constraint (10d), if it were to be added to (PW), is maximally violated under the solution z. In general, this is a hard nonlinear integer programming problem. However, for cutting off the solution z, it suffices merely to find an $(r, \bar{b})$ such that $g(z, r, \bar{b}) > 0$. We next show that this is indeed always possible whenever z is not a cyclic schedule solution.

We denote $T = \text{supp}(z)$. Let Q denote the states "visited" by an optimal solution z to (PW):

$$Q = \bigcup_{(x,a)\in T} (\{x\} \cup \{s(x,a)\}).$$

We include both the starting and ending states because the current z solution may violate flow balance, i.e., may not correspond with a cyclic schedule. If

$$\max_{x' \in Q} \Bigg| \sum_{a\colon (x',a)\in T} z_{x',a} - \sum_{\substack{(\tilde{x},\tilde{a})\in T\\ s(\tilde{x},\tilde{a})=x'}} z_{\tilde{x},\tilde{a}} \Bigg| > 0,$$

then some flow balance constraint of (PC) is violated. Let $\tilde{x}$ be a most offending state, assuming that the above maximum is positive.

Definition 2. The mapping $r\tilde{x}$, for $\tilde{x} \in Q$, is *unique* if no other states in Q map to the same value; i.e., $r\tilde{x} \ne rx$ for every $x \in Q \setminus \{\tilde{x}\}$.

Suppose there exists a j such that $r^j\tilde{x}$ is unique and not already a breakpoint in b^j. Let $i \in [m_j] \cup \{0\}$ index the closest breakpoint to the left of $r^j\tilde{x}$, where by construction breakpoint b^j_0 is guaranteed to exist. Let $\alpha = \max\{r^j x \mid x \in Q, r^j x < r^j\tilde{x}\}$. If α is not defined, then we set $\alpha = -\Omega$. In addition, let $\zeta = \min\{r^j x \mid x \in Q, r^j x > r^j\tilde{x}\}$. If ζ is not defined, then we set $\zeta = \Omega$. Let us select at most three new breakpoints as $\bar{b}^j_1 = \max\{\alpha, b^j_i\}$, $\bar{b}^j_2 = r^j\tilde{x}$, $\bar{b}^j_3 = \min\{\zeta, b^j_{i+1}\}$; see Figure 4. The new constraint (10d) associated with breakpoint $\bar{b}^j_2$ reads as

$$\begin{aligned} 0 &= \sum_{(x,a)\in T} H^j_i(r^j s(x,a)) z_{x,a} - \sum_{(x,a)\in T} H^j_i(r^j x) z_{x,a} \\ &= \sum_{(x,a)\in T\colon s(x,a)=\tilde{x}} H^j_i(r^j s(x,a)) z_{x,a} - \sum_{(x,a)\in T\colon x=\tilde{x}} H^j_i(r^j x) z_{x,a} \\ &= \sum_{\substack{(\tilde{x},\tilde{a})\in T\\ s(\tilde{x},\tilde{a})=\tilde{x}}} z_{\tilde{x},\tilde{a}} - \sum_{a\colon (\tilde{x},a)\in T} z_{\tilde{x},a}, \end{aligned}$$

which is the corresponding flow balance constraint (25c) of (PC). The second equality holds because we constructed the new hat function H^j_i so that the only state for which $r^j x$ evaluates to a nonzero value is $\tilde{x}$. The third equality holds because the hat function is centered about $r^j\tilde{x}$, i.e., equals one there. Because this constraint is violated by z, this proves that the new constraint in (10d) cuts off the solution z.

Suppose for all j that $r^j\tilde{x}$ is already a breakpoint in b^j. If it is unique for some j, then at least one of $\bar{b}^j_1$ and $\bar{b}^j_3$ must be new, thereby producing a new hat function that cuts off z. Otherwise, the constraint (10d) corresponding with the hat function $H_{\bar{b}}$

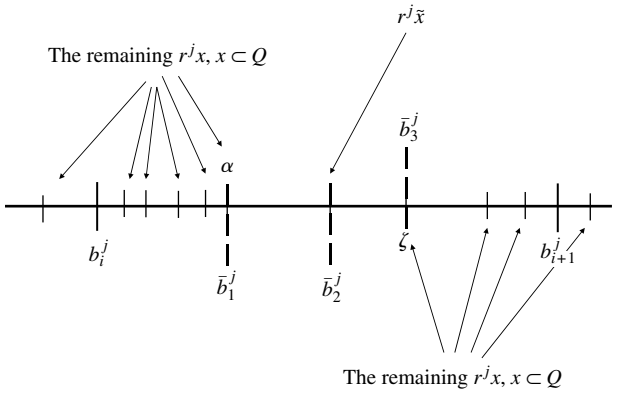

Figure 4 Adding New Breakpoints

must already exist, but be violated, contradicting the feasibility of z.

Finally, we consider the case in which for all $j \in [J]$ the mapping $r^j\tilde{x}$ is not unique. In this case we have to introduce a new ridge vector. We want to find a vector $r = r^{J+1}$ such that the mapping $r\tilde{x}$ is unique. We construct it iteratively as follows. Suppose that we have such an r for a subset $L \subset Q$ and let $\bar{x} \in Q\backslash(L \cup \{\tilde{x}\})$. If $r\tilde{x} \neq r\bar{x}$, then we simply keep the same r and set $L = L \cup \{\bar{x}\}$. Now let $r\bar{x} = r\tilde{x}$. There exists an index $l \in [q]$ such that $\bar{x}_l \neq \tilde{x}_l$. Let

$$\vartheta = 1 + \max_{\substack{x \in L \\ x_l \neq \tilde{x}_l}} \left| \frac{r\tilde{x} - rx}{x_l - \tilde{x}_l} \right|.$$

By definition, $\vartheta \geq 1$. We claim that $r = r + \vartheta e_l$ has the desired property.

We need to show that $(r + \vartheta e_l)x \neq (r + \vartheta e_l)\tilde{x}$ for every $x \in L \cup \{\bar{x}\}$. If $x = \bar{x}$, then this holds because $r\bar{x} = r\tilde{x}$ by definition of l. Now let $x \in L$. If $x_l = \tilde{x}_l$, then the claim holds because of $r\tilde{x} \neq rx'$ for every $x' \in L$. Now let $x_l \neq \tilde{x}_l$, and let us assume that $(r + \vartheta e_l)x = (r + \vartheta e_l)\tilde{x}$. Then

$$\frac{r\tilde{x} - rx}{x_l - \tilde{x}_l} = \vartheta \geq 1 + \left| \frac{r\tilde{x} - rx}{x_l - \tilde{x}_l} \right|,$$

which is a contradiction.

At the end, we scale the resulting r to have the infinity norm of 1. Because the mapping $r\tilde{x}$ is unique, we can implement the above breakpoint generation procedure to produce three breakpoints defining a violated constraint (10d) that is equivalent to the flow balance constraint (25c) for $\tilde{x}$. The above analysis proves the following theorem.

Theorem 1. *Consider an optimal solution z to (PW). If it corresponds with a cyclic schedule, then it is optimal to GJR. Otherwise, we can construct an $(r, \bar{b})$, which cuts off z when the corresponding hat function is added to (PW).*

Note that after adding the new hat function the breakpoints of the existing hat functions are adjusted accordingly, so that b_i^j is strictly increasing in i. Another way of stating this theorem is that if z is feasible to (PW) for a set of ridge vectors and breakpoints, and $g(z, r, \bar{b}) = 0$ for every r and $\bar{b}$, then z is optimal to (PC). This is a variant of the converse of Proposition 3, establishing in essence an equivalence between (PC) and the problem of finding the supremum of $\min(PW)$ over all collections $\{(r^j, b^j)\}_j$.

Example 4. Under the affine value function approximation, an optimal solution to (PW) given previously is

$$z_{\langle 0,0\rangle,\langle 2,3\rangle} = 1/3, \qquad z_{\langle 0,1\rangle,\langle 2,0\rangle} = 1/6.$$

In this case, $Q = \{\langle 0,0\rangle, \langle 1,0\rangle, \langle 0,1\rangle\}$, and we can calculate the left-hand sides of (25c) for each $x \in Q$ as follows: for state $\langle 0,0\rangle$ we have $1/3$, for state $\langle 1,0\rangle$ we have $-1/6$, and for state $\langle 0,1\rangle$ we have $1/6 - 1/3 = -1/2$. The latter is the largest, and hence $\tilde{x} = \langle 0,1\rangle$ is the most offending state. We now search for a ridge vector r such that $r\tilde{x}$ is a unique mapping.

Suppose that the unit vectors $r^1 = \langle 1,0\rangle$ and $r^2 = \langle 0,1\rangle$ are already present, with breakpoints $b^1 = \langle -\epsilon, 0, 2, 2+\epsilon\rangle$ and $b^2 = \langle -\epsilon, 0, 3, 3+\epsilon\rangle$ for any $\epsilon > 0$. As discussed at the end of §2.2.2, this (r, b) collection is redundant for (PW)–(DW) beyond the affine term already present. The set of mappings rx that arise under the ridge vector $r^2 = \langle 0,1\rangle$ over all $x \in Q$ is $\{0,1\}$, and furthermore, only $x = \tilde{x}$ maps into 1, and so it is a unique mapping. Suppose we add a breakpoint to b^2 at 1 so it becomes $b^2 = \langle -\epsilon, 0, 1, 3, 3+\epsilon\rangle$. This gives rise to an additional hat function centered around 1 and in effect a piecewise-linear function with a single breakpoint at 1. After re-solving (DW), we obtain a new optimal solution $\theta = 0$, $v_A = 17.5$, $v_B = 21.667$, $w_1^1 = 0$, $w_2^1 = -15$, $w_3^1 = -15$, to produce the value function approximation

$$u(x) \approx \begin{cases} -17.5x_A - 6.667x_B \\ \quad \text{if } 0 \leq x_B \leq 1, \\ 15 - 17.5x_A - 21.667x_B \\ \quad \text{if } 1 \leq x_B \leq 3, \end{cases} \qquad x = \langle x_A, x_B\rangle \in X,$$

$$\rho = 39.167.$$

Because the four-step cyclic schedule given previously has a cost rate of 39.167, it is optimal for this instance of GJR.

Once a collection (r, b) of ridge vectors and breakpoints is found that makes $\max(DW) = \max(D)$ and $\min(PW) = \inf(PC) = \min(P)$, this does not guarantee that an optimal solution z to (PW) is a cyclic schedule solution. There may be alternative optimal solutions that are not cyclic schedule solutions. Additional breakpoints may still be needed to cut off these alternative optima. This behavior has implications for implementation because it indicates that the z solution may not be relied upon to produce an optimal cyclic schedule even though the optimal objective value is tight with $\min(P)$. More generally, although the (r, b)-generation procedure is guaranteed to cut off the current z solution, it may do so without improving the optimal objective value if there are many alternative optima to (PW). We next illustrate these points on our numerical example.

Example 5. After the breakpoint is added, an optimal solution to (PW) is

$$z_{\langle 0,0\rangle,\langle 2,3\rangle} = 1/6, \quad z_{\langle 0,1\rangle,\langle 2,0\rangle} = 1/3, \quad z_{\langle 2,0\rangle,\langle 0,3\rangle} = 1/6.$$

This can be verified by checking feasibility and complementary slackness. Under this solution, we have $Q = \{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 2,0\rangle\}$ with net flow imbalances of $\{1/6, 0, -1/3, 1/6\}$, respectively, componentwise. Observe that flow balance around state $\langle 0,1\rangle$ is preserved, thanks to the breakpoint that was just added. However, the most offending state is now $\langle 1,0\rangle$. A unique mapping is obtained with $r^1 = \langle 1,0\rangle$; i.e., state $\tilde{x} = \langle 1,0\rangle$ is the only $x \in Q$ that maps into 1 under ridge function r^1. Therefore, we can add a breakpoint at 1 to b^1, which yields $b_1 = \langle -\epsilon, 0, 1, 2, 2+\epsilon\rangle$. (DW) then yields an approximation to $u(x)$ for $x = \langle x^A, x^B\rangle$ that decomposes into $u(x) \approx f^A(x^A) + f^B(x^B)$, pulling the linear terms inside the functions, where

$$f^A(x_A) = \begin{cases} -17.5x_A & \text{if } 0 \le x_A \le 1, \\ 15 - 32.5x_A & \text{if } 1 \le x_A \le 2, \end{cases}$$

and

$$f^B(x_B) = \begin{cases} -6.667x_B & \text{if } 0 \le x_B \le 1, \\ 15 - 21.667x_B & \text{if } 1 \le x_B \le 3. \end{cases}$$

Once flow balance around state $\langle 1,0\rangle$ is coerced, the solver returns as optimal the four-step cyclic schedule solution given in Example 3. This solution is, of course, an alternative optimal solution to (PW) even with only the first breakpoint added, but adding the second breakpoint eliminates the above alternative optimal solution and coerces the solver to produce a cyclic z solution.

5. Algorithm

Putting together the above pieces, we obtain Algorithm 2. Counting iterations with index k, we begin with an initial collection of ridge vectors and associated breakpoints denoted by S^0. Fixing this collection, we then solve (DW) using row generation, or (PW) using column generation, to obtain an optimal solution z^k and associated value function approximation u^k according to (8) obtained from optimal-dual prices. If z^k is a cyclic schedule solution, then according to Theorem 1 it is optimal to GJR. Otherwise, we simulate the approximate policy using Algorithm 1 to obtain its cost rate $\mathscr{B}(u^k)$, which is exact if a cyclic schedule is obtained, and approximate otherwise. If the gap between the upper bound obtained from the policy and the lower bound $\min(PW)$ is below some given threshold ϵ, then the algorithm stops with an optimal or approximate ϵ-optimal schedule. Otherwise, we cut off the solution z^k from (PW) by generating a new hat function, specified by the ridge vector r, and set of three breakpoints $\bar{b}$. Recall that if the ridge vector r already exists and equals r^j for some j, then the set of breakpoints b^j is re-sorted to incorporate the breakpoints $\bar{b}$. We then increase the iteration counter, and the procedure repeats.

Algorithm 2 (The final algorithm)

1: $k = 0$. S^0 denotes an initial collection $\{(r^j, b^j)\}_j$.
2: **loop**
3: Solve (DW) and (PW) with S^k to obtain a solution pair (z^k, u^k).
4: **if** z^k is a cyclic schedule solution **then**
5: Stop. Optimal policy found.
6: **end if**
7: Simulate by applying Algorithm 1 to obtain $\mathscr{B}(u^k)$.
8: **if** $\mathscr{B}(u^k)/\min(PW) < \epsilon$ **then**
9: Stop. Optimal or approximate ϵ-optimal policy found.
10: **end if**
11: $S^{k+1} = S^k \cup \{(r, \bar{b})\}$ where $(r, \bar{b})$ cuts z^k off from (PW)
12: $k = k + 1$
13: **end loop**

Whereas the algorithm given in §4.2 is theoretically guaranteed to cut off the z solution, our numerical experience suggests that it often will do so without improving the lower bound given by $\min(PW)$. This is due to the presence of alternative optima, as illustrated by Example 5. After attempting numerous alternative approaches, it became clear that several properties were important for an (r, b)-generation algorithm to be effective.

First, we want to add as few breakpoints as possible. This is because we need to solve piecewise-linear optimization problems in the solution and policy subproblems, and these are instantiated as mixed-integer programming problems with a quadratic (linear) objective function. Fewer breakpoints means smaller, easier-to-solve instances.

Second, adding breakpoints to the unit ridge vectors, which essentially amounts to adding breakpoints to the separable piecewise-linear component, improved the objective function more frequently and by a larger amount than other ridge vectors. Therefore, in our final implementation we give preference to them.

Third, we want breakpoints that regain flow balance around many states at once. The algorithm described in §4.2 isolates the most offending state $\tilde{x} \in Q$ and cuts off the z solution by adding constraints that (loosely) enforce flow balance around $\tilde{x}$. However, Klabjan and Adelman (2007) prove convergence for an algorithm solving (P) when (26) is solved to generate a new ridge vector and breakpoints. In general, the objective function g will be maximized by a hat function into and out of which

multiple states flow under the z solution. For this reason, and our desire for fewer breakpoints, we only add the center breakpoints $\bar{b}_2^j$ and not the other (new) breakpoints $\bar{b}_1^j$ and $\bar{b}_3^j$. Theoretically, it is possible that adding only the center breakpoint will not cut, if multiple flow imbalances cancel each other out. Practically speaking, however, we find it does cut.

We finally settled on the following heuristic approach, shown in Algorithm 3, with F being a parameter. It first tries to add a breakpoint to unit ridge vectors, and if this fails it tries to add a breakpoint to one of the existing other ridge vectors. If this fails, then a new ridge vector and associated breakpoints are generated.

For each $x' \in Q$, let

$$f_{x'} = \left| \sum_{a:(x',a)\in T} z_{x',a} - \sum_{\substack{(\bar{x},\bar{a})\in T \\ s(\bar{x},\bar{a})=x'}} z_{\bar{x},\bar{a}} \right|$$

denote the flow imbalance of state x'. Also, let E denote the subset of ridge vector indices $[J]$ corresponding with unit vectors in $\mathbb{R}^{\mathcal{I}}$. Let us assume that $f_{x^{\langle 1\rangle}} \geq f_{x^{\langle 2\rangle}} \geq \cdots \geq f_{x^{\langle |Q|\rangle}}$, where $Q = \{x^{\langle 1\rangle}, x^{\langle 2\rangle}, \ldots, x^{\langle |Q|\rangle}\}$. Last, let

$$U_d^{\bar{J}} = \{j \in \bar{J}\colon r^j x^{\langle d\rangle} \text{ is a unique mapping}\}$$

denote the set of ridge vector (indices) under which $x^{\langle d\rangle}$ generates a unique mapping, restricted to a subset $\bar{J} \subseteq [J]$. These are the ridge vectors eligible for adding breakpoints.

Algorithm 3 ((r, b)-Generation algorithm to cut off z by coercing flow balance among at most F states)

1: Calculate Q, f_x for all $x \in Q$, and sort Q by largest f_x first.
2: **for** $d = 1, \ldots, \min\{F, |Q|\}$ **do**
3: **if** $U_d^E \neq \varnothing$ **then**
4: Choose $j \in U_d^E$ uniformly at random.
5: $b^j \leftarrow b^j \cup \{r^j x^{\langle d\rangle}\}$
6: **else if** $U_d^{[J]\setminus E} \neq \varnothing$ **then**
7: Choose $j \in U_d^{[J]\setminus E}$ uniformly at random.
8: $b^j \leftarrow b^j \cup \{r^j x^{\langle d\rangle}\}$
9: **else**
10: Generate a new ridge vector r^{J+1} and breakpoints b^{J+1} such that $r^{J+1}x^{\langle d\rangle}$ is a unique mapping.
11: $J \leftarrow J + 1$
12: **end if**
13: **end for**

In Steps 3–5 of Algorithm 3, we try to add a new breakpoint based on an existing unit vector. Similarly, if this fails, in Steps 6–8 we attempt to introduce a new breakpoint to an existing nonunit ridge vector. If all this fails, then we generate a new ridge vector as follows. Choose any subset $\tilde{Q} \subseteq Q$ of states to regain flow balance around. In the context of Algorithm 3, in Step 10 we set $\tilde{Q} = \{x^{\langle d\rangle}\}$ for a single d. The idea is to choose an (r, b) so that breakpoints are as spread out as possible, indirectly forcing unique mappings when possible. The following program maximizes the sum of the minimum distances between mappings rx:

$$\max_{r:\|r\|_\infty \leq 1} \sum_{\tilde{x}\in\tilde{Q}} \min_{x\in Q\setminus\{\tilde{x}\}} |rx - r\tilde{x}|.$$

This can be written as a linear complementarity problem:

$$\max_{r,\theta,\alpha,\beta} \sum_{\tilde{x}\in\tilde{Q}} \theta_{\tilde{x}}$$

$$\theta_{\tilde{x}} \leq \alpha_{x,\tilde{x}} + \beta_{x,\tilde{x}} \quad \tilde{x} \in \tilde{Q},\ x \in Q\setminus\{\tilde{x}\},$$

$$rx - r\tilde{x} = \alpha_{x,\tilde{x}} - \beta_{x,\tilde{x}} \quad \tilde{x} \in \tilde{Q},\ x \in Q\setminus\{\tilde{x}\},$$

$$-1 \leq r_i \leq 1 \quad i \in \mathcal{I},$$

$$\alpha_{x,\tilde{x}}\beta_{x,\tilde{x}} = 0 \tilde{x} \in \tilde{Q},\ x \in Q\setminus\{\tilde{x}\}.$$

The breakpoints can be taken to be the ones to the left, center, and right of $r\tilde{x}$, as described in §4.2, for every $\tilde{x} \in \tilde{Q}$. The algorithm in §4.2 generates a feasible solution to this program with a positive objective value, which proves that a ridge vector that is optimal to this program, together with associated breakpoints, is guaranteed to cut off the z solution. In practice, we rarely need to generate new ridge vectors by solving this complementarity problem; i.e., Steps 10 and 11 are seldom executed. This is because we start with a large initial collection that includes all unit vectors, and all "pair vectors," which set all components of ridge vector r to zero except $r_{i_1} = 1/\lambda_{i_1}$ and $r_{i_2} = -1/\lambda_{i_2}$ for $i_1, i_2 \in \mathcal{I}$ such that $i_1 < i_2$. This implies that $rx = x_{i_1}/\lambda_{i_1} - x_{i_2}/\lambda_{i_2}$, which calculates the difference in stockout times between items i_1 and i_2, given current inventories.

Given these initial ridge vectors, we initialize breakpoints as follows. For each j, we set $b_1^j = \min_{x\in X} r^j x$, $b_2^j = \max_{x\in X} r^j x$, and $m_j = 2$. Then we choose any $b_0^j < b_1^j$ and any $b_3^j > b_2^j$. This corresponds with two hat functions, one centered at the leftmost point of the domain and the other centered at the rightmost point of the domain. Proposition 1 shows that this corresponds with an affine function, and so it adds no additional fidelity on top of the affine term $\theta - vx$ already in the approximation (8). In this case, the constraints (10d) are redundant and can be dropped without any loss. When we add a breakpoint to b^j for some j between b_1^j and b_2^j, we obtain three hat functions, but this effectively models a piecewise-linear function with only one breakpoint in the interior of the domain. When we report the number of

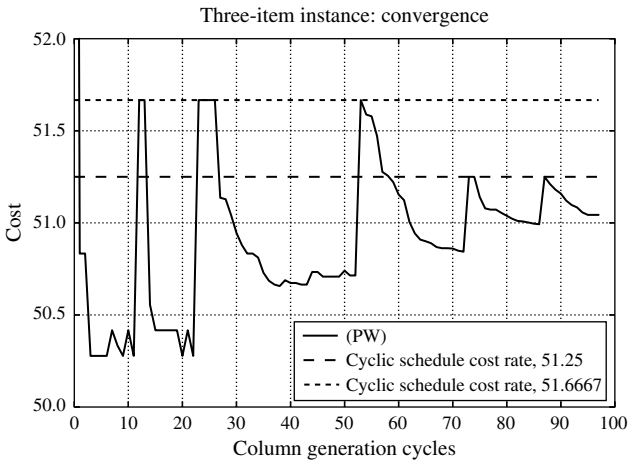

Figure 5 Numerical Convergence of Upper and Lower Bounds

breakpoints, we provide the number excluding the two leftmost and two rightmost breakpoints of b^j, i.e., $m_j - 2$, so that in Example 2 we effectively have only one breakpoint.

Figure 5 demonstrates how the algorithm typically converges, on a three-item instance. Every time we solve the column generation subproblem for (PW), we report the new objective value of the restricted master problem version of (PW) with the new column added. Whenever the graph reaches the bottom of a trough, we obtain the optimal objective value of (PW) with the current ridge vectors and breakpoints. The curve typically jumps up again once new breakpoints are added. As can be observed, the lower bounds given by (PW) improve. Furthermore, the simulation yields two cyclic schedules, one with value 51.667 and one with value 51.25. The algorithm terminates with a guarantee that the latter schedule is within 0.5% of the optimal value. In Figure 6 we show how the maximum flow imbalance across all states visited, i.e., $\max_{x\in Q}|f_x|$, decreases as breakpoints are added, on an instance with 10 items.

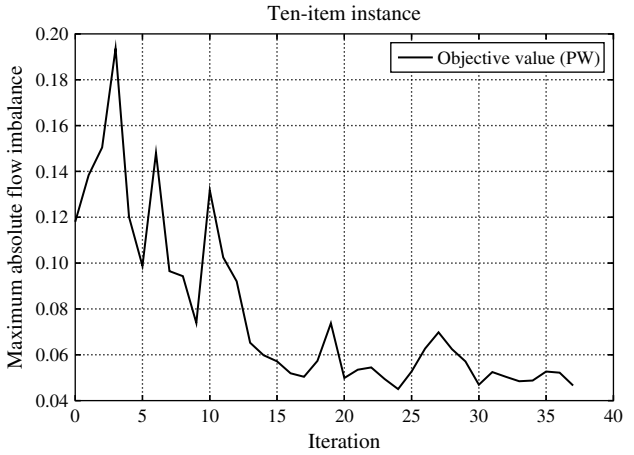

Figure 6 Numerical Convergence of Flow Imbalances

6. Computational Results

We generated two collections of instances. Table 1 depicts results from a collection of 18 problem instances, designed to mimic the real-world data as in Adelman (2003). In particular, for item i we take $\lambda_i/\overline{X}_i$ to be exponentially distributed and the storage limits $\overline{X}_i$ to be uniformly distributed, which fit the real-world data well. After sampling these two distributions, we then compute the implied λ_i. We generated instances having 5, 10, and 15 items, with six of each set. The first three in each set have $\bar{A}$ equal to the sum of the first two-thirds' smallest storage limits $\overline{X}_i$, whereas the last three effectively set $\bar{A} = \infty$ so that replenishment capacity is not constraining. We then vary the holding cost between 0 (none), 1 (small), and 5 (large). We report results for major/minor fixed costs, which simplifies computation, with the major cost equal to 100 and minor costs uniformly distributed between 0 and 60. We also ran instances with traveling salesman costs but found that the results were not materially different.

Table 2 presents additional results for similar instances, except without holding costs and varying the structure of item storage capacities. "Random" means that for each i, $\overline{X}_i = 10\lambda_i U_i + \lambda_i$, where U_i represents a sample for item i from the uniform distribution over the real interval $[0, 1]$. Hence, with an inventory level of $\overline{X}_i$, item i will stock out in one time unit plus a uniformly distributed time interval between 0 and 10. "Constant" means that $\overline{X}_i = \overline{X}$ for all i, where the constant $\overline{X} = \sum_{j\in\mathcal{F}} \lambda_j U_j + \sum_{j\in\mathcal{F}} \lambda_j/|\mathcal{F}|$. "Discrete" means that $\overline{X}_i = \alpha_i \overline{X}$, where $\alpha_i \in \{2, 4, 8\}$ with probability 1/3 each. Finally, each λ_i is taken to be uniformly distributed on the line segment $[0, 10]$. All aforementioned random variables are resampled independently for each of the 42 instances depicted in this table.

Because the simulation is computationally intensive, we terminate the algorithm when the best simulated schedule is within 2% of optimal, or after 48 hours, whichever comes sooner. We also set the look-ahead parameter $\mathcal{N}$ to three or four periods. The column "Total capacity" represents $\bar{A}$ expressed as a fraction of items' storage capacities covered. The column entitled "Initial LB" reports the optimal objective value given by (PW) with only the affine value function approximation, which gives a lower bound. The column entitled "Final LB" reports the optimal objective value of (PW) when the algorithm terminates. We also report how many instances of (PW) we solve, along with the number of breakpoints (#b) added between unit vectors and pair vectors. On these instances, the algorithm never needed to produce new ridge vectors, except for instances 1 and 5 in Table 2, in which two and three ridge vectors, respectively, are generated. We also report the best

Table 1 Numerical Results on Instances with Varying Holding Costs

Instance	No. of items	Holding cost	Total capacity	Initial LB	Final LB	No. of LPs	#b unit	#b pair	Best UB	No. of steps	Opt. FP	FP/UB	LB final /initial	UB/LB
1	5	0	2/3	14.12	14.87	2	2	0	14.90	220	16.40	1.101	1.054	1.002
2	5	1	2/3	260.91	260.91	1	0	0	262.45	2	266.71	1.016	1.000	1.006
3	5	5	2/3	325.88	325.88	1	0	0	326.19	2	334.41	1.025	1.000	1.001
4	5	0	1	11.21	12.53	2	4	0	12.77	30	13.84	1.084	1.118	1.020
5	5	1	1	270.85	270.85	1	0	0	271.73	2	278.34	1.024	1.000	1.003
6	5	5	1	255.79	255.79	1	0	0	255.83	2	258.15	1.009	1.000	1.000
7	10	0	2/3	24.33	25.70	32	63	2	26.28	1,000	33.70	1.282	1.056	1.023
8	10	1	2/3	419.95	419.95	1	0	0	424.97	30	440.25	1.036	1.000	1.012
9	10	5	2/3	924.44	924.44	1	0	0	932.20	30	968.24	1.039	1.000	1.008
10	10	0	1	21.43	21.62	3	4	0	22.01	1,000	25.87	1.176	1.009	1.018
11	10	1	1	408.24	410.00	5	8	0	418.16	88	436.93	1.045	1.004	1.020
12	10	5	1	881.58	886.72	2	8	0	897.74	5	950.59	1.059	1.006	1.012
13	15	0	2/3	20.57	23.43	2	7	1	23.65	1,000	27.56	1.165	1.139	1.009
14	15	1	2/3	472.30	472.30	1	0	0	474.72	12	498.98	1.051	1.000	1.005
15	15	5	2/3	1,239.71	1,239.71	1	0	0	1,283.33	48	1,313.85	1.024	1.000	1.035
16	15	0	1	36.77	41.26	29	59	0	42.12	1,000	52.31	1.242	1.122	1.021
17	15	0	1	26.51	27.40	21	43	0	28.12	1,000	35.97	1.279	1.034	1.026
18	15	5	1	1,064.32	1,068.87	2	4	0	1,129.98	397	1,137.57	1.007	1.004	1.057

Note. #b, number of breakpoints.

upper bound obtained through simulation, including the number of steps in the corresponding schedule. We find empirically that a cyclic schedule is produced by all instances in Table 1 except those reporting 1,000 or more steps and all instances in Table 2 reporting under 200 steps. In these cases, the upper-bound value is exact. In the other instances, the upper-bound value is approximate because we truncated the simulation when either 1,000 (or 4,000) periods are reached or based on a convergence criterion that uses running average cost.

As a reference point, under column "Opt. FP," we report the objective value of an optimal fixed-partition policy (FP) (Rosenblatt and Kaspi 1985, Queyranne 1987, Goyal 1987). We calculate this by solving the set-partitioning problem

$$\min \sum_{I \subseteq \mathcal{I}} g_I Y_I$$

$$\sum_{I \subseteq \mathcal{I}: i \in I} Y_I = 1 \quad i \in \mathcal{I},$$

$$Y_I \text{ binary} \quad I \subseteq \mathcal{I},$$

where g_I is the average cost of the policy that replenishes all items $i \in I$ together. This is easy to calculate by computing the optimal time between replenishments,

$$t_I^* = \min \left\{ \sqrt{2 \cdot C_I / \sum_{i \in I} h_i \lambda_i}, \bar{A} / \sum_{i \in I} \lambda_i, \min_{i \in I} \bar{X}_i / \lambda_i \right\},$$

if $\sum_{i \in I} h_i > 0$; otherwise, we drop the first term in the minimum. Then

$$g_I = \frac{C_I}{t_I^*} + t_I^* \sum_{i \in I} \frac{h_i \lambda_i}{2}.$$

After reporting the optimal FP value, we then report the following ratios: "FP/UB" reports the optimal FP value divided by the value of best simulated policy, "LB final/initial" reports the improvement in the lower bound achieved by adding breakpoints and new ridge vectors, and last "UB/LB" reports the value of the simulated policy divided by the lower bound and thus gives an optimality guarantee. (We solve the lower bound to within 0.1% of optimality, and so this guarantee is approximate.)

Several observations are in order:

1. The lower bound can be improved up to 14% by adding breakpoints.

2. Usually the unit vectors suffice for adding breakpoints, but occasionally the pairs are needed. We can almost always successfully add a breakpoint to unit vectors and pairs; i.e., there is rarely a need to generate additional ridge vectors.

3. We add more than 50 breakpoints on instances with 15 items and up to 82 breakpoints on instances with 6 items.

4. We beat the optimal fixed-partition policy consistently and substantially by as much as 28%.

5. Sometimes long cyclic schedules are detected in the simulation.

6. The best simulated policy usually performs within 2% of optimality, and usually this is the policy that is produced by the affine value function approximation, i.e., without any breakpoints. This may be largely due to looking ahead, because it becomes less important to fit the true value function exactly when applied further out in time. It suggests that the main advantage of ridge generation is in improving the lower bound but that it is not needed to obtain a

Table 2 Numerical Results on Instances Without Holding Costs but Varying Item Storage Capacities

Instance	No. of items	Total capacity (%)	Item capacities	Initial LB	Final LB	No. of LPs	#b unit	#b pair	Best UB	No. of steps	Opt. FP	FP/UB	LB final /initial	UB/LB
1	4	50	Random	116.31	117.28	40	41	37	119.48	330	126.76	1.061	1.008	1.019
2	4	100	Random	26.66	28.05	2	1	1	28.50	1	28.50	1.000	1.052	1.016
3	4	50	Constant	68.50	68.50	1	0	0	69.36	71	72.60	1.047	1.000	1.013
4	4	100	Constant	142.72	142.72	1	0	0	143.37	63	160.11	1.117	1.000	1.005
5	4	50	Discrete	387.43	387.88	42	47	35	397.69	33	423.05	1.064	1.001	1.025
6	4	100	Discrete	102.03	108.56	2	2	0	109.87	13	112.63	1.025	1.064	1.012
7	6	33	Random	186.88	186.88	1	0	0	187.66	657	194.16	1.035	1.000	1.004
8	6	67	Random	149.31	149.31	1	0	0	150.15	220	189.20	1.260	1.000	1.006
9	6	100	Random	65.05	70.91	4	4	0	72.07	3	82.09	1.139	1.090	1.016
10	6	33	Constant	188.29	188.29	42	48	34	200.65	408	212.52	1.059	1.000	1.066
11	6	67	Constant	102.34	102.34	3	0	4	119.59	28	120.56	1.008	1.000	1.169
12	6	100	Constant	99.94	99.94	1	0	0	101.46	9	125.50	1.237	1.000	1.015
13	6	33	Discrete	263.59	268.31	2	2	0	270.86	457	278.11	1.027	1.018	1.009
14	6	67	Discrete	319.96	352.69	20	30	8	358.35	237	373.99	1.044	1.102	1.016
15	6	100	Discrete	128.75	143.31	7	12	0	145.17	20	170.92	1.177	1.113	1.013
16	8	25	Random	698.96	698.96	1	0	0	700.01	4,000	722.32	1.032	1.000	1.002
17	8	50	Random	218.08	219.44	37	48	26	226.83	470	261.72	1.154	1.006	1.034
18	8	75	Random	172.20	178.84	20	33	5	179.88	228	222.36	1.236	1.039	1.006
19	8	100	Random	178.53	186.34	20	31	7	188.98	20	231.90	1.227	1.044	1.014
20	8	25	Constant	114.75	114.75	28	27	27	117.24	480	123.87	1.057	1.000	1.022
21	8	50	Constant	63.64	63.64	5	3	5	69.80	169	77.66	1.113	1.000	1.097
22	8	75	Constant	69.30	71.08	13	20	4	72.26	24	90.73	1.256	1.026	1.017
23	8	100	Constant	95.14	107.99	9	16	0	108.12	80	114.14	1.056	1.135	1.001
24	8	25	Discrete	445.95	445.95	1	0	0	454.19	715	468.92	1.032	1.000	1.018
25	8	50	Discrete	289.51	320.02	13	23	1	330.34	291	396.28	1.200	1.105	1.032
26	8	75	Discrete	131.27	134.99	5	7	1	137.34	210	160.71	1.170	1.028	1.017
27	8	100	Discrete	144.63	163.75	10	13	5	166.72	24	197.00	1.182	1.132	1.018
28	10	20	Random	703.08	703.08	1	0	0	703.48	4,000	703.36	1.000	1.000	1.001
29	10	40	Random	278.60	278.60	1	0	0	279.60	4,000	294.61	1.054	1.000	1.004
30	10	60	Random	159.42	159.42	26	46	4	163.76	236	213.12	1.301	1.000	1.027
31	10	80	Random	115.71	116.59	15	22	6	123.85	270	148.79	1.201	1.008	1.062
32	10	100	Random	158.65	162.01	7	11	1	166.66	60	191.21	1.147	1.021	1.029
33	10	20	Constant	114.53	114.53	1	0	0	115.62	493	119.83	1.036	1.000	1.010
34	10	40	Constant	79.02	79.02	1	0	0	85.41	21	88.23	1.033	1.000	1.081
35	10	60	Constant	58.42	66.47	16	25	5	67.44	40	77.94	1.156	1.138	1.015
36	10	80	Constant	84.84	96.14	31	44	16	100.25	68	104.76	1.045	1.133	1.043
37	10	100	Constant	109.72	122.76	10	16	2	125.15	234	145.56	1.163	1.119	1.019
38	10	20	Discrete	461.66	461.66	1	0	0	461.57	4,000	471.23	1.021	1.000	1.000
39	10	40	Discrete	430.49	433.17	31	54	6	453.90	515	474.52	1.045	1.006	1.048
40	10	60	Discrete	500.36	500.36	3	0	4	551.13	264	677.45	1.229	1.000	1.101
41	10	80	Discrete	251.95	258.72	3	4	0	263.38	222	326.67	1.240	1.027	1.018
42	10	100	Discrete	242.55	261.59	27	47	5	268.50	252	341.72	1.273	1.078	1.026

Note. #b, number of breakpoints.

strong policy. This is good news for scalability in practice, because it means that breakpoints can be avoided if one is only interested in policies and not their performance guarantee. Adelman (2003) solves instances without holding cost having up to 134 items; this is done by exploiting a problem structure that could be investigated in future work for the case with holding costs.

7. There does not seem to be a discernible pattern of difference in the performance of the policy nor the bounds as problem instance parameters change.

In the major/minor cost setting, the partitioning algorithm is one of the best-known algorithms for constructing steady-state solutions. Our algorithm beats this heuristic on a regular basis. Up until this work, the best-known lower bound was obtained by linear value function approximation. Again, our algorithm, by using more general functions, significantly improves these lower bounds for most of the instances. We observed that unit and pairwise ridge vectors are beneficial, but other ridge vectors are of limited use. We firmly believe that such more-general vectors could further improve the algorithm, but they have to be selected judiciously. Toward this end, the key is to study alternatives to the complementarity problem introduced in §5.

7. Concluding Remarks

Although we have not explored the efficacy of our methodology on problems other than the GJR prob-

lem, it is more broadly applicable. As described in Klabjan and Adelman (2006) and Hernández-Lerma and Lasserre (1996), the basic infinite-dimensional math programming framework summarized in §2.1.2 can be applied to any semi-Markov decision process defined on Borel state and action spaces. This includes problems with stochasticity in their transition law and discrete spaces. It also applies to discrete-time problems by setting the transition times $\tau(x, a) = 1$. The works cited above provide conditions to check when verifying strong duality and/or the existence of stationary optimal policies.

For the linear programming approach to approximate dynamic programming to work fully as described in this paper, one must be able to solve the separation problem (11), either heuristically or optimally. When there is stochasticity, we must compute the expected future value relative to the value function approximation. The viability of this depends on problem structure; e.g., Adelman (2004) reports success on a multi-item stochastic inventory control problem with discrete state and action spaces. If the separation problem can be solved, then the policy problem shown in Algorithm 1 can be solved, because it is easier as a result of fixing the initial state. In cases for which the separation problem cannot be solved, researchers have considered constraint sampling methods (e.g., de Farias and Van Roy 2004).

Our methodology for basis generation is promising for problems for which good policies exist on a relatively small set of positive recurrent states, and especially when they are optimal or ϵ-optimal. For deterministic problems, such policies correspond with finite cyclic schedules. However, the methodology still applies when the kernel is stochastic but permits transition into a given state only from a finite set of previous states. For such problems, we are able to effectively exploit the connection between (PC) and (PW), as articulated in Proposition 3 and Theorem 1. In this stochastic case, (PC) has an expectation in (25c) but is still computable as a finite sum. For any state $\tilde{x} \in Q$ that violates flow balance (25c), a new hat function can be found that isolates this discrete point at the middle breakpoint and separates the corresponding z solution from the problem (PW). However, if finding a good policy requires Q to grow infinitely large, then the number of breakpoints needed would become unwieldy; if Q approaches a continuous set, then other states may map arbitrarily closely to the middle breakpoint. This happens when good policies require a continuous or infinite set of positive recurrent states. Further research is needed to identify other specific problems, such as the GJR problem, for which underlying finite substructure makes our methodology effective.

References

Adelman, D. 2003. Price-directed replenishment of subsets: Methodology and its application to inventory routing. *Manufacturing Service Oper. Management* **5**(4) 348–371.

Adelman, D. 2004. A price-directed approach to stochastic inventory/routing. *Oper. Res.* **52**(4) 499–514.

Adelman, D., D. Klabjan. 2005. Duality and existence of optimal policies in generalized joint replenishment. *Math. Oper. Res.* **30**(1) 28–50.

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows*. Prentice Hall, Englewood Cliffs, NJ.

Croxton, K. L., B. Gendron, T. L. Magnanti. 2003. A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems. *Management Sci.* **49**(9) 1268–1273.

de Farias, D. P., B. Van Roy. 2003. The linear programming approach to approximate dynamic programming. *Oper. Res.* **51**(6) 850–865.

de Farias, D. P., B. Van Roy. 2004. On constraint sampling in the linear programming approach to approximate dynamic programming. *Math. Oper. Res.* **29**(3) 462–478.

Dror, M. 2005. Routing propane deliveries. A. Langevin, D. Riopel, eds. *Logistics Systems: Design and Optimization*. Springer, New York, 299–322.

Goyal, S. K. 1987. Comment on "A dynamic programming approach for joint replenishment under general order cost functions." *Management Sci.* **33**(1) 133–135.

Hernández-Lerma, O., J. B. Lasserre. 1996. *Discrete-Time Markov Control Processes: Basic Optimality Criteria*. Springer-Verlag, Berlin.

Klabjan, D., D. Adelman. 2006. Existence of optimal policies for semi-Markov decision processes using duality for infinite linear programming. *SIAM J. Control Optim.* **44**(6) 2104–4122.

Klabjan, D., D. Adelman. 2007. An infinite-dimensional linear programming algorithm for deterministic semi-Markov decision processes on Borel spaces. *Math. Oper. Res.* **32**(3) 528–550.

Powell, W. B. 2007. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons, New York.

Queyranne, M. 1987. Comment on "A dynamic programming algorithm for joint replenishment under general order cost functions." *Management Sci.* **33**(1) 131–133.

Rosenblatt, M. J., M. Kaspi. 1985. A dynamic programming algorithm for joint replenishment under general order cost functions. *Management Sci.* **31**(3) 369–373.

Schweitzer, P. J., A. Seidmann. 1985. Generalized polynomial approximations in Markovian decision processes. *J. Math. Anal. Appl.* **110**(2) 568–582.