

Simulation-Based Optimization of Virtual Nesting Controls for Network Revenue Management

Garrett van Ryzin

Graduate School of Business, Columbia University, New York, New York 10027,
gjr1@columbia.edu

Gustavo Vulcano

Stern School of Business, New York University, New York, New York 10012,
gvulcano@stern.nyu.edu

Virtual nesting is a popular capacity control strategy in network revenue management. In virtual nesting, products (itinerary-fare-class combinations) are mapped (“indexed”) into a relatively small number of “virtual classes” on each resource (flight leg) of the network. Nested protection levels are then used to control the availability of these virtual classes; specifically, a product request is accepted if and only if its corresponding virtual class is available on each resource required. Bertsimas and de Boer proposed an innovative simulation-based optimization method for computing protection levels in a virtual nesting control scheme [Bertsimas, D., S. de Boer. 2005. Simulation-based booking-limits for airline revenue management. *Oper. Res.* 53 90–106]. In contrast to traditional heuristic methods, this simulation approach captures the true network revenues generated by virtual nesting controls. However, because it is based on a discrete model of capacity and demand, the method has both computational and theoretical limitations. In particular, it uses first-difference estimates, which are computationally complex to calculate exactly. These gradient estimates are then used in a steepest-ascent-type algorithm, which, for discrete problems, has no guarantee of convergence.

In this paper, we analyze a continuous model of the problem that retains most of the desirable features of the Bertsimas-de Boer method, yet avoids many of its pitfalls. Because our model is continuous, we are able to compute gradients exactly using a simple and efficient recursion. Indeed, our gradient estimates are often an order of magnitude faster to compute than first-difference estimates, which is an important practical feature given that simulation-based optimization is computationally intensive. In addition, because our model results in a smooth optimization problem, we are able to prove that stochastic gradient methods are at least locally convergent. On several test problems using realistic networks, the method is fast and produces significant performance improvements relative to the protection levels produced by heuristic virtual nesting schemes. These results suggest it has good practical potential.

Subject classifications: stochastic algorithm; stochastic gradients; revenue management; capacity control.

Area of review: Manufacturing, Service, and Supply Chain Operations.

History: Received January 2003; revisions received September 2004, September 2005, February 2006, May 2006; accepted September 2006.

Introduction

Optimally rationing the amount of capacity sold to various demand classes is a central problem in airline revenue management (RM). The so-called *single-resource problem* involves rationing capacity on a single resource (single flight leg or single day at a hotel property). This problem has received significant attention in the academic literature in the past and a variety of heuristic and exact methods have been proposed for single-resource problems; see, for example, Littlewood (1972), Belobaba (1987, 1989), Brumelle and McGill (1993), and Lee and Hersh (1993). The book by Talluri and van Ryzin (2004) provides a detailed discussion of single-resource methods.

More recently, RM practitioners and researchers have turned their attention to network, or origin-destination (O-D), RM. In network RM, the objective is to jointly manage the capacities of a network of related resources. For exam-

ple, network problems arise in managing the capacities of a set of flights in a hub-and-spoke airline network with connecting and local traffic, or in managing hotel capacity on consecutive days based on guests' lengths of stay. The dependence among the resources in these cases is created by customer demand; customers may require several resources simultaneously (e.g., two connecting flights, or a number of consecutive days at a hotel) to satisfy their needs. Thus, accepting demand will reduce capacity on multiple resources simultaneously. To make network-optimal revenue decisions in such cases requires a network-level approach to capacity control; that is, optimization methods that explicitly model network effects and control mechanisms that determine product availability based on total network revenue contribution. (See Talluri and van Ryzin 2004 for an in-depth discussion of network RM methods.)

Although network RM creates both methodological and implementation challenges, the potential revenue benefits are significant. Indeed, simulation studies of airline hub-and-spoke networks by various researchers have demonstrated significant revenue gains from using network methods over single-resource methods; see, e.g., Belobaba and Lee (2000), Belobaba (2001), and Williamson (1992). Several major airlines are now using network methods and most vendors now offer some form of network RM system.

Yet, exact network optimization is, practically speaking, impossible due to the large dimensionality of the resulting dynamic programming models. Instead, various approximate methods have been developed. Some approximations are based on deterministic mathematical programming models. Among these are the minimum-cost network-flow formulations by Glover et al. (1982) and Dror et al. (1988). Williamson (1992) proposed and investigated mathematical programming methods based on linear and nonlinear programming approximations. Curry (1990) describes a combined mathematical programming and nested allocation formulation of the network problem, in which fare classes within the same O-D are nested, while capacity is partitioned among O-Ds.

Other approximations are based on decomposing the network problem into a collection of single-resource problems. One of the oldest decomposition approximations is the so-called *virtual nesting method*, developed initially at American Airlines (Smith and Penn 1988, Smith et al. 1992). In virtual nesting, products (itinerary-fare-class combinations) are clustered according to some criteria (usually an estimate of the net network revenue benefit of selling the product) to form a small number of “virtual classes” on each resource of the network. This mapping of products to virtual classes is called *indexing*. Demand and revenue statistics from these aggregated virtual classes are then used as inputs to a single-resource model, which provides nested protection levels (or booking limits) for the virtual classes on each resource. A product is then open for sale if and only if capacity is available in its corresponding virtual class on each of the resources it requires. In this way, the network problem is decomposed into a collection of single-resource problems. Yet, by clustering itineraries into a relatively small number of classes based on their estimated network revenue values, virtual nesting is able to approximate the “network effects” of selling a product. Moreover, the approach preserves the nested protection level structure of traditional single-resource controls—an important advantage in practice because often the reservation system is designed only to implement resource-level, class-based booking controls. For these reasons, virtual nesting methods have proved to be quite popular in practice. (Again, see Talluri and van Ryzin 2004 for a complete discussion of virtual nesting methods.)

Recently, an interesting variation on virtual nesting was proposed by Bertsimas and de Boer (2005). Their idea is to view the mapping of products to virtual classes and the

collection of nested protection levels as defining a “class” of control strategies, parameterized by the set of protection levels on the network. Starting with a given initial set of protection levels, they then use simulation-based optimization and approximate dynamic programming methods to optimize these protection levels. The key idea is that by using simulation, it is possible to provide a very accurate estimate of the true network effects of changing protection levels. This is a significant advantage relative to traditional virtual nesting methods, which are based on single-resource models that, at best, only heuristically approximate network effects.

However, to perform this simulation-based optimization, Bertsimas and de Boer (2005) use a discrete-capacity, discrete-demand model of the network problem. They divide the booking horizon into time periods, and when the booking process enters a new time period, the protection levels are reoptimized. This is accomplished by using simulation-based optimization to determine the revenue generated during the current time period and by using an approximation of the value function to estimate the revenue-to-go from the next period onward. While running the simulation, if a protection level becomes binding, it is perturbed and the resulting revenue is estimated by spawning a new data structure to keep track of the two sample paths (perturbed and unperturbed). This is repeated for every binding protection level encountered.

Although this discrete model is realistic, it leads to a difficult optimization problem in several respects. For one, first-difference estimates can be computationally inefficient. Indeed, as mentioned, to compute all first-differences requires generating a separate sample path for each binding protection level encountered. This can quickly become too computationally intensive to be practical for large networks. (A detailed complexity analysis is provided in §2.4.) This is one reason Bertsimas and de Boer (2005) break the booking horizon into periods and use an approximation of the revenue-to-go rather than directly simulating the complete sample path. In fairness, their revenue-to-go approximation also has the desirable feature of reoptimizing the protection level parameters in each period, so it further serves as a value function estimate in the spirit of approximate dynamic programming. We do not consider this sort of value function approximation here; rather, our focus is on the simulation-based optimization portion of their approach. However, it is certainly possible to combine our more efficient simulation method with Bertsimas and de Boer’s value function approximation approach; the two ideas are quite independent.

A second difficulty is that Bertsimas and de Boer (2005) use their first-difference gradient estimates in a stochastic steepest ascent algorithm, together with rounding at each iteration. The drawback here is that ascent methods together with rounding at each step are not guaranteed to converge for discrete problems. Even algorithms based on finite first-differences do not necessarily converge, either locally or

globally. (See Ermoliev 1988.) In short, the method lacks good convergence properties.

Despite these computational and theoretical limitations, Bertsimas and de Boer (2005) report promising numerical results. Moreover, the overall idea of their approach is a clever and practical one.

Motivated by the potential of this approach, we propose and analyze a variation of it that overcomes the shortcomings identified above. The key difference is that we use a continuous capacity and demand (fluid) model of the problem that allows for partial acceptances. This fluid-model approach is similar in spirit to that in Mahajan and van Ryzin (2001) for a stochastic inventory problem. Although less realistic in terms of the fine-grained details of the simulation, the advantage of the model is that it leads to a much easier continuous optimization problem. Indeed, we show how to compute the entire sample path gradient for the network revenue function exactly via a simple recursion that is essentially no more complex than simulating a single sample path. Numerical examples show that our gradient estimates can often be computed an order of magnitude faster than first-difference estimates. The recursion is also quite easy to implement. The pseudocode is provided in online Appendix A4. An electronic companion to this paper is available as part of the online version that can be found at <http://or.journal.informs.org/>.

We then embed these sample path gradients in a stochastic approximation algorithm as in Bertsimas and de Boer's (2005) method, but without rounding at each iteration. Because our revenue function is sufficiently smooth, we are able to prove that it is locally convergent. (As a final heuristic step, one can always round the final parameter values as needed to generate integer-valued protection levels.) Our numerical experiments show significant revenue increases with respect to the starting protection levels using our method. Moreover, it is quite fast even on problems of realistic size.

We note that both speed and performance are important factors in RM practice. Indeed, simulation-based optimization is notoriously computationally intensive, and in large airline implementations of network RM, one must often solve many instances of large network problems in a small processing time window. For example, a major network carrier can easily have networks consisting of thousands of flight legs and hundreds of thousands of products, and they may need to optimize one such network for each day going out 300 days into the future. All this optimization is typically done in an overnight batch process along with data extraction, forecasting, reservation system updating, and other computationally intensive tasks. The total available time window for optimization may be only on the order of a few hours, corresponding to a few minutes per network in most cases. Indeed, one reason simple deterministic models remain so popular in practice is precisely because they are extremely fast to solve. Therefore, if simulation-based methods are to become viable alternatives in practice, computational efficiency is essential.

The remainder of this paper is organized as follows. In §1, we introduce the discrete model and its continuous approximation. Section 2 describes the sample path view of the network demand. In §3, we present the way we improve an initial set of protection levels through a gradient-based method. Section 4 shows some numerical results, and we present our conclusions in §5.

Notation

We begin by introducing some notational conventions. For a vector x , x_j denotes its j th component, and x^T is the vector transpose. For a number a , we denote $a^+ = \max\{a, 0\}$. Let \mathcal{N} denote the set $\{1, \dots, n\}$. We use $\mathbf{I}\{\cdot\}$ for the indicator function, a.s. means *almost surely*, c.d.f. is short for *cumulative distribution function*, and w.p.1 is short for *with probability 1*.

1. Model Formulation

The network has m resources which can be used to provide n products (e.g., in an airline network each resource could correspond to a single-leg flight, and a product is defined by an itinerary and price-class combination). Define the incidence matrix $A = [a_{ij}] \in \{0, 1\}^{m \times n}$. We let $a_{ij} = 1$ if resource i is used by product j , and $a_{ij} = 0$ otherwise. Thus, the j th column of A , A_j , is the incidence vector for product j ; and the i th row of A , A^i , is the incidence vector for resource i . We use the notation $i \in A_j$ to indicate that resource i is used by product j ; and $j \in A^i$ to mean that product j uses resource i . The state of the network is described by a vector $x^T = (x_1, \dots, x_m)$ of resource capacities. If one unit of product j is sold, the state of the network changes to $x - A_j$. To simplify the analysis, we will ignore cancellations and no-shows. The revenue obtained from accepting a request for one unit of product j is r_j .

A request for product j is mapped to virtual class $c_i(j)$ on each resource i used by product j as given by a fixed indexing scheme, which we assume is given. A variety of methods can be used to perform this indexing (we discuss a standard one in online Appendix A3) and our algorithm works with any of them. Although some indexing schemes will no doubt yield better performance than others, the indexing scheme per se is not the focus of our analysis; rather, it is assumed to be an input to our model.

We assume that there are $\bar{c} + 1$ virtual classes on each resource, and that virtual class 1 is the highest in the nesting order, followed by virtual class 2, etc. Let y_{ic} denote the protection level for virtual classes c and higher on resource i . Under a virtual nesting control, requests mapped to virtual class $c + 1$ are accepted if and only if the remaining capacity on each leg i required by the booking exceeds y_{ic} (the protection level for higher virtual classes). In other words, class $c + 1$ requests only have access to the capacity in excess of y_{ic} on leg i .

Let $y = (y_{11}, \dots, y_{1\bar{c}}, \dots, y_{m1}, \dots, y_{m\bar{c}})$ denote the vector of all $m\bar{c}$ protection levels. The protection levels are nested, so we require

$$0 \leq y_{i1} \leq y_{i2} \leq \dots \leq y_{i\bar{c}} \leq x_i(1), \quad i = 1, \dots, m, \quad (1)$$

where $x_i(1)$ is the initial capacity of resource i . Without loss of generality, we assume that $x_i(1) > 0 \forall i$, else we can simply eliminate leg i and all products $j \in A^i$ from the model formulation. Let Θ be the set of all y satisfying these constraints. We assume dummy protection levels y_{i0} when needed: $y_{i0} = 0 \forall i$, representing the fact that there is no protection level for the highest virtual class on each leg.

We assume that the protection levels y remain constant while processing the requests. This corresponds to what is called *theft nesting* (de Boer 2003). Another alternative, called *standard nesting*, is to update protection levels after requests are accepted. Specifically, in standard nesting, if a request for one seat is accepted for a virtual class c on a given leg, the protection levels for classes c and lower (i.e., classes $c, c+1, \dots, \bar{c}$) are reduced by one. The rationale for this adjustment is heuristic, but the idea is roughly to compensate for lower expected demand-to-come for class c after we observe an arrival of a class c request. There is effectively no difference between these two versions of nesting when low-fare demand arrives before high-fare demand. However, if the order of arrivals is mixed (as is possible in virtual nesting schemes), theft nesting can potentially “overprotect” capacity for high-fare demand, although the relative performance of the two methods can depend on the frequency of reoptimization of the protection levels. While both nesting methods are found in airline industry practice, we restrict our analysis here to theft nesting, although our approach can be modified to work with standard nesting.

We take a sample path view of the demand and sales process. Let T denote the number of customers on a sample path, verifying $\mathbb{P}(T \leq \tau) = 1$ for some finite constant τ . The demand is characterized as a sequence of customer requests $\omega = \{\delta_t : t = 1, \dots, T\}$. The index t runs forward in time (i.e., $t = 1$ represents the first customer, $t = 2$ the second customer, and $t = T$ the last one). Each element δ_t in the sequence is a pair $\delta_t = (j_t, q_t)$, where j_t is a realization of a random variable on the support \mathcal{N} representing the product type requested by customer t , and q_t is a realization of a random variable with support $[0, \bar{Q}]$, denoting the amount requested. While T, j_t , and q_t are all functions of ω , we omit this dependence explicitly to prevent the notation from becoming too cumbersome.

To ensure the expected revenue function is smooth, in our theoretical analysis we will require that the quantities q_t have a continuous differentiable c.d.f. $F_q(\cdot)$, although it is clearly more realistic to view q_t as discrete (e.g., the number of seats). In our computations, we tested this more realistic discrete-demand case.

Note that this demand model is quite general. It allows arbitrary order of arrivals among the virtual classes, arbitrary correlation and coefficients of variation between successive demands, etc. This level of generality is one of the main advantages of simulation-based methods; essentially, any procedure for generating sequences of demand of this form is allowed.

Our key assumption is that capacity and demand are continuous quantities and that requests can be partially accepted. Specifically, a request t for an amount $q_t = q$ of product $j_t = j$ is processed as follows: By letting $x(t)$ denote the vector of remaining capacities at time t , the amount of request accepted $u_j(x(t), y, q)$ is given by the minimum available capacity among all the resources required by j , or q if there is at least q units of capacity on all these resources. Formally,

$$u_j(x(t), y, q) = \min\{q, (x_i(t) - y_{i, c_i(j)-1})^+ : i \in A_j\}. \quad (2)$$

Essentially, we are considering demands as fluids, which we can partially accept if the available capacity is positive but less than the quantity q requested.

For the same sample path ω , define $R_t(x(t), y, \omega)$ to be the revenue-to-go over periods $t, t+1, \dots, T$ starting with a vector $x(t)$ of remaining capacities and protection levels y . We then have the following set of recursive *forward equations* for determining the revenues:

$$R_t(x(t), y, \omega) = r_{j_t} u_{j_t}(x(t), y, q_t) + R_{t+1}(x(t+1), y, \omega),$$

$$x(t+1) = x(t) - A_{j_t} u_{j_t}(x(t), y, q_t)$$

for $t = 1, \dots, T$, with boundary conditions $R_{T+1}(x, y, \omega) = 0$ for all x, y, ω . The total sample path revenue is given by

$$R(y, \omega) = R_1(x(1), y, \omega). \quad (3)$$

Our objective is to maximize the expected revenue over the set Θ of feasible protection levels:

$$\max_{y \in \Theta} g(y), \quad \text{where} \quad (4)$$

$$g(y) = E[R(y, \omega)].$$

(Here, and in what follows, expectation is taken with respect to the random sequence of demand.)

2. Sample Path Analysis

We begin by analyzing the sample path revenue $R_t(x(t), y, \omega)$ as a function of the protection levels y and the remaining capacity $x(t)$.

2.1. Smoothing the Revenue Function

Note that by allowing partial acceptance of requests, the function $u_j(x(t), y, q)$ defined by (2) is continuous and

piecewise linear in y . Moreover, one can easily verify that $y_{i, c_i(j)-1} = x_i(t) - q$ and $y_{i, c_i(j)-1} = x_i(t)$ are points of non-differentiability, which makes $R(y, \omega)$ a continuous but nonsmooth function of y . Indeed, we cannot even guarantee that $R_t(x(t), y, \omega)$ is differentiable with respect to y w.p.1, because the event $y_{i, c_i(j)-1} = x_i(t)$ can occur with some positive probability (e.g., with positive probability, we can get a sequence of high-quantity requests such that the value $u_j(x(t), y, q) = 0$ in (2) for a sequence of consecutive t s is determined by the fact that $y_{i, c_i(j)-1} = x_i(t)$). This fact violates the well-known sufficient conditions for the differentiability of $g(y)$, and in particular for interchanging differentiation and expectation. (See Glasserman 1994 for a good reference on this topic.)

To overcome these technical difficulties, we redefine the sample path ω as $\omega = \{\delta_t: t = 1, \dots, T\}$, where now $\delta_t = (j_t, q_t, \zeta_t)$, and where $\zeta_{t,i}$ is a random variable uniformly distributed on $[0, \epsilon]$ for some small ϵ .¹ Then, we consider the following variation of the problem:

$$R_t(x(t), y, \omega) = r_{j_t} u_{j_t}(x(t) - \zeta_t, y, q_t) + R_{t+1}(x(t+1), y, \omega), \quad (5)$$

$$x(t+1) = x(t) - \zeta_t - A_{j_t} u_{j_t}(x(t) - \zeta_t, y, q_t). \quad (6)$$

The idea here is to “smooth” the acceptance function by randomly perturbing the remaining capacity. Indeed, with this new perturbation, following the argument in the previous paragraph, the event $y_{i, c_i(j)-1} = x_i(t) - \zeta_{t,i}$ occurs with probability zero. Thus, the control $u_j(\cdot)$ defined in (2) becomes differentiable w.p.1. Using the composition defined by (5), it is not hard to see that the revenue function becomes differentiable w.p.1. as well.

We are now able to calculate the gradient of the revenue function with respect to y . From (3), we have

$$\begin{aligned} \nabla_y R(y, \omega) &= \nabla_y R_1(x, y, \omega) \\ &= \left(\frac{\partial}{\partial y_{11}} R_1(x, y, \omega), \dots, \frac{\partial}{\partial y_{m\bar{c}}} R_1(x, y, \omega) \right). \end{aligned}$$

Analogously, the gradient with respect to x is

$$\nabla_x R_1(x, y, \omega) = \left(\frac{\partial}{\partial x_1} R_1(x, y, \omega), \dots, \frac{\partial}{\partial x_m} R_1(x, y, \omega) \right).$$

Using the chain rule, we then obtain the set of *backward equations* for the derivatives with respect to y_{ic} :

$$\begin{aligned} &\frac{\partial}{\partial y_{ic}} R_t(x(t), y, \omega) \\ &= \left(r_{j_t} - \sum_{k \in A_{j_t}} \frac{\partial}{\partial x_k(t+1)} R_{t+1}(x(t+1), y, \omega) \right) \\ &\quad \cdot \frac{\partial}{\partial y_{ic}} u_{j_t}(x(t) - \zeta_t, y, q_t) + \frac{\partial}{\partial y_{ic}} R_{t+1}(x(t+1), y, \omega) \\ &\quad \forall i, c, t. \quad (7) \end{aligned}$$

A similar set of backward equations for the derivatives with respect to x_i is given by

$$\begin{aligned} &\frac{\partial}{\partial x_i(t)} R_t(x(t), y, \omega) \\ &= \left(r_{j_t} - \sum_{k \in A_{j_t}} \frac{\partial}{\partial x_k(t+1)} R_{t+1}(x(t+1), y, \omega) \right) \\ &\quad \cdot \frac{\partial}{\partial x_i(t)} u_{j_t}(x(t) - \zeta_t, y, q_t) \\ &\quad + \frac{\partial}{\partial x_i(t+1)} R_{t+1}(x(t+1), y, \omega) \quad \forall i, t, \quad (8) \end{aligned}$$

with boundary conditions

$$\begin{aligned} &\frac{\partial}{\partial y_{ic}} R_{T+1}(x(T+1), y, \omega) = 0 \quad \forall i, c, \\ &\frac{\partial}{\partial x_i(T+1)} R_{T+1}(x(T+1), y, \omega) = 0 \quad \forall i. \end{aligned}$$

The online appendix provides a detailed derivation of these derivatives of the revenue function.

Note that the general form of the two derivatives is very similar. The term in parentheses is simply the marginal revenue for accepting one extra unit of product j minus the marginal displacement cost over the legs used by product j —in other words, product j ’s displacement adjusted revenue value. This quantity is multiplied by the derivatives of the acceptance function ($(\partial/\partial y_{ic})u_j(x, y, q)$ or $(\partial/\partial x_i)u_j(x, y, q)$) to give the marginal value in the current period. Adding this to the marginal revenue-to-go gives the total derivative.

2.2. Gradients of u_j

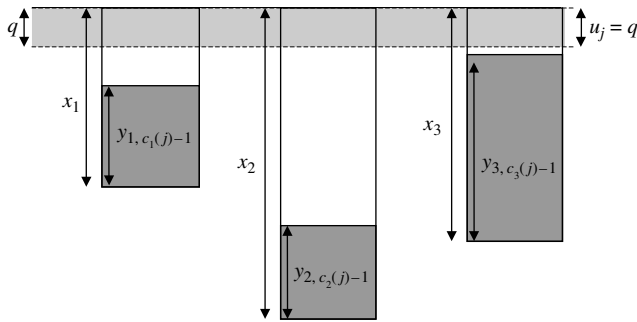
We next determine the gradients of the acceptance function $u_j(x, y, q)$. To ensure that partial derivatives are well defined on the boundary of the feasible set (1), we redefine (2) as follows:

$$u_j(x, y, q) = \min\{q, (x_i - y_{ic})^+ : i \in A_j, c < c_i(j)\}. \quad (9)$$

From (9), one can determine for all i and c the following derivative:

$$\frac{\partial}{\partial y_{ic}} u_j(x, y, q) = \begin{cases} -1 & \text{if simultaneously} \\ & \text{(i) } i \in A_j, \\ & \text{(ii) } x_i - y_{ic} < x_k - y_{k,c'} \\ & \quad \forall k \in A_j, k \neq i, c' < c_k(j), \\ & \text{(iii) } 0 < x_i - y_{ic} < q, \\ & \text{(iv) } c < c_i(j), \\ \neq & \text{if (ii) or (iii) holds at equality,} \\ 0 & \text{otherwise.} \end{cases} \quad (10)$$

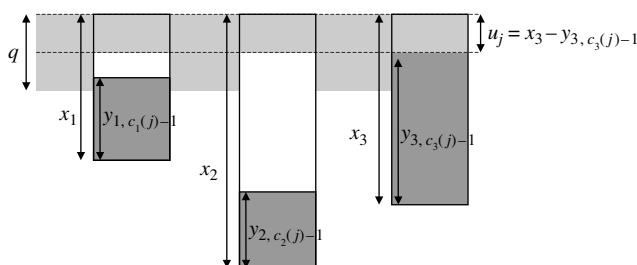
Figure 1. Stochastic gradient calculation of the acceptance function with respect to nested protection levels: The zero gradient case.



In words, the quantity of demand accepted from a request for product j in state x is reduced (one-for-one) by a slight increase in the protection level y_{ic} if and only if all of the following hold: (i) resource i is used by j , (ii) the capacity available on resource i is a binding constraint, (iii) the amount accepted is positive but constrained by the protection level associated to product j over resource i , and (iv) class c is higher in the nesting order than the virtual class of product j over resource i . If at least one equality holds for (ii) or (iii), then the derivative does not exist. However, from (7), one can see that the probability of this event occurring is zero due to the random perturbation of capacity, and the continuity of demand quantity q_i . In all other cases, a small change in y_{ic} does not affect the amount of j we accept.

These conditions are further illustrated in Figures 1 and 2, which show a request at time t for q units of product j that uses resources 1, 2, and 3. The height of the bars represent the capacity remaining at time t , and the quantities $y_{i, c_i(j)-1}$ represent the protection levels for product j on each resource i . The unshaded areas therefore represent the capacity available for product j on each of the three resources. Note in Figure 1 that there is sufficient capacity available on all three resources to fully satisfy the request for product j , so $u_j = q$. Thus, a small perturbation in $y_{i, c_i(j)-1}$ will not affect the quantity of product j accepted in this time period, and therefore $(\partial/\partial y_{i, c_i(j)-1})u_j(x, y, q) = 0$

Figure 2. Stochastic gradient calculation of the acceptance function with respect to nested protection levels: The nonzero gradient case.



for all i . However, in Figure 2, the requested quantity exceeds the available capacity on resource 3 and the request can only be partially filled because of the protection level constraint on resource 3, so $u_j = x_3 - y_{3, c_3(j)-1}$. In this case, a small increase in $y_{3, c_3(j)-1}$ will reduce the amount of product j that we accept in this period, so $(\partial/\partial y_{3, c_3(j)-1})u_j(x, y, q) = -1$. An increase in any of the other protection levels on resources 1 and 2, however, will not affect the quantity accepted because these protection levels are not binding. This example illustrates the conditions leading to (10).

A similar reasoning provides the derivatives with respect to x_i :

$$\frac{\partial}{\partial x_i} u_j(x, y, q) = \begin{cases} 1 & \text{if simultaneously} \\ & \text{(i) } i \in A_j, \\ & \text{(ii) } x_{ic} - y_{ic} < x_k - y_{k, c'} \\ & \quad k \in A_j, k \neq i, c' < c_k(j) \\ & \quad \text{for some } c < c_i(j), \\ & \text{(iii) } 0 < x_i - y_{ic} < q \\ & \quad \text{for some } c < c_i(j), \\ \bar{\exists} & \text{if (ii) or (iii) holds at equality,} \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

In words, the quantity of demand accepted from a request for product j in state x is decreased (one-for-one) by a slight decrease in the capacities x_i if and only if all of the following hold: (i) resource i is used by j , (ii) the capacity available on resource i is a binding constraint, and (iii) there is some positive capacity available on resource i , but constrained by the protection level associated to product j over resource i (or by one from a higher virtual class). Again, the derivative does not exist if (ii) or (iii) holds with equality, but as before, because of the perturbation of capacity and the continuity of the distribution of q , this event occurs with probability zero. In all other cases, a small change in x_i does not affect the amount of j we accept.

2.3. Example of Revenue Gradient

The following example illustrates the mechanics of the gradient calculation: Consider a single-leg problem, with three products, three virtual classes (one per product), and revenues $r = (25, 19, 10)$. Suppose that the initial capacity is $x(1) = 8$, and that protection levels are $y = (2, 4)$. Assume for simplicity that all requests are for a single seat, and suppose that the sequence of requests for products is $(3, 3, 3, 3, 2, 1)$, e.g., corresponding to a sample path

$$\omega = \{(3, 1, 0.01), (3, 1, 0.006), (3, 1, 0.003), (3, 1, 0.001), (2, 1, 0.03), (1, 1, 0.007)\}.$$

In this case, the fourth request for product 3 meets a binding constraint (e.g., after the realization of the random variable $\zeta_4 = 0.001$, $x(4) - \zeta_4 = 4.98$) because four seats are reserved for classes 2 and 1. Then, when marginally increasing y_2 (or decreasing x), we will be rejecting a marginal unit of product 3, decreasing the revenue by 10. The gradients here are

$$\nabla_y R_1(x(1), y, \omega) = (0, -10) \quad \text{and} \\ \frac{\partial}{\partial x(1)} R_1(x(1), y, \omega) = 10.$$

Now, take the same initial situation, and suppose instead that the sequence of product requests is (3, 3, 3, 3, 2, 2, 2, 1, 1, 1). Here, when processing the second request for product 2, according to Equation (6), the remaining capacity will be a value close to but less than three, say $x(6) - \zeta_6 = 2.993$. In other words, the second request for product 2 meets a binding constraint and is not fully accepted because there are two seats reserved for class 1. The second seat for class 1 is not fully accepted either because there is no more space available at this point. The gradients here are then

$$\nabla_y R_1(x(1), y, \omega) = (6, 9) \quad \text{and} \quad \frac{\partial}{\partial x(1)} R_1(x(1), y, \omega) = 10.$$

In words, by incrementing y_2 , we will be able to accept an additional increment of product 2 at the expense of reducing a marginal unit of product 3. So, $(\partial/\partial y_2)R_1(x(1), y, \omega) = 19 - 10$. Similarly, by incrementing y_1 , we will accept an additional increment of product 1, but lose an increment of product 2: $(\partial/\partial y_1)R_1(x(1), y, \omega) = 25 - 19$. Regarding the sensitivity with respect to $x(1)$, when fixing y and decreasing the number of seats available, we reduce the amount of product 3 accepted.

2.4. Complexity Analysis

The procedure for calculating a sample-based gradient is described in detail in online Appendix A4. It basically consists of two passes: In the *forward pass*, we simulate the acceptance decisions and keep track of the state (available capacity) of the network, $x(t)$, observed by each arriving customer t . The *backward pass* then rebuilds the capacity seen by each customer, identifies all the binding protection levels, and calculates the gradient using (7) and (8). The overall complexity of this routine is determined by the backward pass.

Let K denote the total number of binding protection levels encountered on the sample path, and recall that T denotes the total number of customers. Then, the computation of the sample path gradient takes $O(K + T)$ time.² The finite-difference method of Bertsimas and de Boer (2005), in contrast, takes $O(K \times T)$ for the same sample because an efficient implementation of their algorithm consists of two forward passes: In the first pass, we identify all the

binding protection levels (again, there are K). In the second pass, for each binding protection level, we increase the protection level by one and keep track of the acceptance decisions and revenues from that request onward under the increased protection level. This requires tracking another $O(T)$ booking requests and simulating the acceptance decisions under the new protection level. This results in the complexity of $O(K \times T)$ cited above. For realistic size networks under high load factors, one can expect large values of K , and hence the complexity difference can become quite significant.

The above complexity is based on worst-case analysis. To give a sense of the real-world difference in speed between these two gradient estimates, we performed a series of numerical tests on two sample networks. Table 1 reports the CPU time taken by both the first-difference and our continuous gradient estimate over the same 100 sample paths for the two different networks.³ As suggested by the theoretical complexity analysis, the difference in computation time becomes more significant when the networks are congested (and hence, more binding protection levels are met along the sample path).⁴

Note that the running time difference between the methods approaches an order of magnitude in the larger network case. But even this network is relatively modest in size compared to large commercial airline networks. Indeed, it would not be unreasonable to expect differences of two orders of magnitude or more in very large airline networks. Such differences in speed can well mean the difference between solving and not solving a network in a practical amount of time.

2.5. Properties of the Revenue Function

We next look at several important theoretical properties of the revenue function.

2.5.1. Quasiconcavity. The important result here is a negative one: namely, we show that, in general, the sample path revenue function is not quasiconcave in the protection

Table 1. Comparison of CPU times between the stochastic gradient and finite-difference methods over 100 sample paths.

Network			Average	Stochastic	Finite-	Magnitude
m	n	Demand factor	sample size	gradient CPU seconds	difference CPU seconds	of time improvement
8	80	1.25	1,030	0.657	2.827	4.3
8	80	1.11	1,030	0.653	2.372	3.6
8	80	1.00	1,030	0.641	1.803	2.8
8	80	0.91	1,030	0.625	1.531	2.4
62	1,844	1.28	7,696	4.870	49.411	10.1
62	1,844	1.10	7,696	4.826	39.783	8.2
62	1,844	0.96	7,696	4.796	35.563	7.4
62	1,844	0.86	7,696	4.897	32.321	6.6
62	1,844	0.77	7,696	4.921	29.423	6.0

Table 2. Sample path for the quasiconcavity counterexample.

Customer t	Product j_t	$y = (3, 5)$		$z = (1, 7)$		$0.5y + 0.5z$	
		Decision	$x(t + 1)$	Decision	$x(t + 1)$	Decision	$x(t + 1)$
1	2	Accepted	7	Accepted	7	Accepted	7
2	3	Accepted	6	Rejected	7	Accepted	6
3	3	Accepted	5	Rejected	7	Rejected	6
4	2	Accepted	4	Accepted	6	Accepted	5
5	2	Accepted	3	Accepted	5	Accepted	4
6	2	Rejected	3	Accepted	4	Accepted	3
7	2	Rejected	3	Accepted	3	Accepted	2
8	2	Rejected	3	Accepted	2	Rejected	2
9	1	Accepted	2	Accepted	1	Accepted	1
10	1	Accepted	1	Accepted	0	Accepted	0
11	1	Accepted	0	Rejected	0	Rejected	0
Total revenue		63		62		61	

levels vector. The significance of this result is that without quasiconcavity, we cannot preclude the possibility that there maybe local optima in the expected revenue of the continuous problem (4). While this is not entirely surprising, this property is worth verifying formally.

THEOREM 1. *There exist sample paths ω on which the sample path revenue function $R(y, \omega)$ for the continuous problem is not quasiconcave.*

PROOF. We exhibit a sample path in Table 2, on which the revenue function is not quasiconcave.⁵ There is a single leg ($m = 1$), and three products (virtual classes) with revenues $r = (10, 7, 6)$. The capacity is set at $x = 8$, and 11 customers arrive, with requests for just one unit each. There are two sets of protection levels given by $y = (3, 5)$, i.e., the seller reserves three seats for the highest class and five for classes 1 and 2; and $z = (1, 7)$. Their convex combination given by $\alpha y + (1 - \alpha)z$ at $\alpha = 0.5$ is the set of protection levels (2, 6). Columns labeled $x(t + 1)$ represent the remaining capacity for the next customer $t + 1$. Given the integrality of protection levels and the unit requests, each customer is fully accepted or rejected.

Table 2 shows that the revenue from protection levels y is $3 \times 10 + 3 \times 7 + 2 \times 6 = 63$ (i.e., the seller has accepted three products 1, three products 2, and two products 3). From protection levels z , revenue is $2 \times 10 + 6 \times 7 = 62$, and from the average protection levels, revenue is $2 \times 10 + 5 \times 7 + 1 \times 6 = 61$. Hence, quasiconcavity is not verified; that is, this sample path has strictly fewer revenue using a convex combination of two sets of protection levels, in this case (2, 6), than with both $y = (3, 5)$ and $z = (1, 7)$. \square

2.5.2. Continuity and Differentiability. We will use the following observations in the sequel (see online Appendix A2). For any $t = 1, \dots, T$, there exist constants k_y and k_x such that

$$\left| \frac{\partial}{\partial y_{ic}} R_t(x(t), y, \omega) \right| \leq k_y, \tag{12}$$

and

$$\left| \frac{\partial}{\partial x_k(t)} R_t(x(t), y, \omega) \right| \leq k_x. \tag{13}$$

The next lemma records a direct consequence of (12): The revenue function is Lipschitz continuous in the protection levels.

LEMMA 1. *Let y and z be two feasible sets of protection levels. Then, there exists a constant K_R , independent of x and ω , such that*

$$|R_t(x(t), z, \omega) - R_t(x(t), y, \omega)| \leq K_R \|z - y\|$$

for all $t = 1, \dots, T$ for the continuous, a.s. differentiable model (5)–(6).

The next lemma justifies the interchange of the expectation and differentiation operators on a sample path ω :

LEMMA 2. *For the randomly perturbed model (5)–(6), the gradient $\nabla_y E[R(y, \omega)]$ exists for all $y \in \Theta$, and $\nabla_y E[R(y, \omega)] = E[\nabla_y R(y, \omega)]$.*

PROOF. By definition of partial derivative,

$$\lim_{h \rightarrow 0} \frac{R(y + h e_{ic}, \omega) - R(y, \omega)}{h} = \frac{\partial}{\partial y_{ic}} R(y, \omega),$$

where e_{ic} denotes the vector in $\mathcal{R}^{m \times \bar{c}}$ that has a one in component i, c , and zero otherwise. Note that the numerator $|R(y + h e_{ic}, \omega) - R(y, \omega)| \leq k_y |h|$, and hence for all sequences $\{h_k\}$ such that $h_k \rightarrow 0$, the random variable $R_{ic}^k(y, \omega) \equiv (R(y + h_k e_{ic}, \omega) - R(y, \omega))/h_k$ verifies $|R_{ic}^k(y, \omega)| \leq k_y$ for all k and ω , and hence it is *uniformly bounded*. Now, applying the bounded convergence theorem (e.g., see Billingsley 1995, Theorem 5.4), we get for all sequences $h_k \rightarrow 0$,

$$\lim_{h_k \rightarrow 0} E[R_{ic}^k(y, \omega)] = E \left[\frac{\partial}{\partial y_{ic}} R(y, \omega) \right].$$

The left-hand side is in fact

$$\frac{\partial}{\partial y_{ic}} E[R(y, \omega)],$$

and the statement of the lemma follows. \square

The following continuity result is critical for solving (4).

THEOREM 2. *The objective function $g(y)$ is continuously differentiable on Θ .*

PROOF. From Lemma 2, $\nabla g(y) = \nabla_y E[R(y, \omega)] = E[\nabla_y R(y, \omega)]$, where $\nabla_y R(y, \omega)$ is defined by the (perturbed) recursion (7). As discussed in §2.1, for any given $\hat{y} \in \Theta$, this recursion is differentiable at \hat{y} w.p.1, and moreover, by the same argument, the gradient is also continuous at \hat{y} w.p.1. That is, for any sequence of vectors $\{y_k\}$ tending to \hat{y} ,

$$\lim_{y_k \rightarrow \hat{y}} \nabla_y R(y_k, \omega) = \nabla_y R(\hat{y}, \omega) \text{ w.p.1.}$$

But from (12), the derivatives with respect to y are uniformly bounded by k_y . Hence, by the bounded convergence theorem and Lemma 2, we have that

$$\begin{aligned} \lim_{y_k \rightarrow \hat{y}} \nabla_y g(y_k) &= \lim_{y_k \rightarrow \hat{y}} E[\nabla_y R(y_k, \omega)] \\ &= E[\nabla_y R(\hat{y}, \omega)] \\ &= \nabla_y g(\hat{y}). \end{aligned}$$

Hence, $\nabla g(y)$ is continuous at y for all $y \in \Theta$. \square

2.5.3. Bounded Variance. The following result shows that the variance of the stochastic partial derivative of the revenue function is bounded. This result is important for proving the local convergence of stochastic gradient methods.

LEMMA 3. *There exists a finite constant C such that $\text{Var}((\partial/\partial y_{ic})R(y, \omega)) \leq C$.*

PROOF. Applying (12),

$$\text{Var}\left(\frac{\partial}{\partial y_{ic}}R(y, \omega)\right) \leq E\left[\left(\frac{\partial}{\partial y_{ic}}R(y, \omega)\right)^2\right] \leq k_y^2.$$

So, it is enough to take $C = k_y$. \square

3. Stochastic Gradient Algorithm

We next look at using the sample path gradients in a simulation-based optimization algorithm. There are two main approaches for doing this: sample average approximation methods (SAA) and stochastic approximation methods (SA).

SAA methods generate a fixed set of sample paths and then solve a deterministic problem on this fixed sample using traditional optimization techniques. The theory and application of SAA are discussed in Plambeck et al. (1996), Kleywegt and Shapiro (2001), and Shapiro (2000). In a nonsmooth framework such as our original formulation in §1, bundle methods like the ones discussed in Lemaréchal (1989, §6) are appropriate.

In contrast, SA methods generate a single sample path per iteration, and then a gradient from this sample is used as

a “steepest ascent” direction. A step is taken in this direction and the procedure is repeated. The stochastic approximation method originated in Robbins and Monro (1951), Kushner and Clark (1978), Benveniste et al. (1990), and the recent book by Kushner and Yin (2003) contain expositions of its theory.

Some applications in the OR/OM field are the queuing papers by L’Ecuyer and Glynn (1994) and L’Ecuyer et al. (1994). Mahajan and van Ryzin (2001) solve by an SA method an inventory problem where customers substitute among product variants within a retail assortment when inventory is depleted. In the RM arena, Karaesmen and van Ryzin (2004) apply these techniques to an overbooking problem with substitutable inventory classes.

A sketch of the stochastic gradient method is as follows: For a random function $F(x) = E_\omega[f(x, \omega)]$, the method attempts to maximize $F(x)$ on a constraint set X through the iterations

$$x^{(k+1)} = \Pi_X(x^{(k)} + \rho^{(k)} s^{(k)}), \quad x^{(0)} \in X, \quad k = 0, 1, \dots,$$

where $\Pi_X(\cdot)$ is a projection operator on the set X , $\rho^{(k)}$ is a step length, and $s^{(k)}$ is an estimate of the gradient of $F(x^{(k)})$.

Based on preliminary tests using both the SAA and the SA methods, we settled on using SA. In our experience, naive versions of the SAA were less efficient than SA because the cost of each gradient calculation was quite high (e.g., SAA required computing gradients for, say, 1,000 sample paths for each iteration versus one in the case of SA). Thus, despite the fact that SA required more iterations in total, it was faster overall. However, this is not an entirely fair comparison because we did not try SAA with more advanced bundle methods, which would likely improve the efficiency of the approach significantly. In general, more sophisticated algorithms applied to SAA would be worth investigating but are somewhat beyond the scope of this paper. Our aim, instead, was simply to implement a straightforward version of the algorithm to get a first-cut sense of its performance. In addition, the SA algorithm is provably convergent in our case (as we show below) and thus it has some theoretical value as well.

3.1. Algorithm and Implementation Choices

To maximize $g(y) = E[R(y, \omega)]$ over the convex compact set Θ defined by constraints (1), we require an initial feasible point $y^{(0)} \in \Theta$, and a sequence of step sizes $\{\rho^{(k)}\}$ satisfying

$$\begin{aligned} \rho^{(k)} > 0, \quad \lim_{k \rightarrow \infty} \rho^{(k)} = 0, \quad \sum_{k=1}^{\infty} \rho^{(k)} = +\infty, \quad \text{and} \\ \sum_{k=1}^{\infty} [\rho^{(k)}]^2 < +\infty. \end{aligned} \tag{14}$$

In particular, we have chosen a step size $\rho^{(k)} = a/k$ for some constant $a > 0$.

For simulated demand streams $\omega^{(1)}, \dots, \omega^{(N)}$, our implementation of the stochastic gradient method proceeds as follows.

Stochastic Gradient Algorithm

Step 1. Compute an initial feasible set of protection levels $y^{(0)}$.

Step 2. For $k := 1$ to N do:

(a) Calculate the sample path gradient over demand stream $\omega^{(k)}$: $\nabla_y R(y^{(k-1)}, \omega^{(k)})$.

(b) Set new step size $\rho^{(k)} := a/k$.

(c) Update the protection levels for the next iteration, using the equation

$$y^{(k)} := \Pi_{\Theta}(y^{(k-1)} + \rho^{(k)} \nabla_y R(y^{(k-1)}, \omega^{(k)})),$$

where $\Pi_{\Theta}(\cdot)$ is the orthogonal projection into the feasible set Θ .

Step 3. Return $y^{(N)}$. Stop.

Some comments about the implementation of this algorithm are in order. We ran a fixed number, N , of iterations to improve an initial feasible set of protection levels obtained in Step 1 (typically, N was on the order of thousands). In practice, various stopping criterion can be employed to terminate the algorithm, although one weakness of stochastic gradient methods is that they lack good stopping criteria (Shapiro 2000).

The step size chosen in Step 2(b) is a popular choice. Alternative step-size rules for more general stochastic quasigradient methods can be found in Pflug (1988).

The projection in Step 2(c) is of the form

$$y = \Pi_{\Theta}(z) \Leftrightarrow y = \arg \min_{y' \in \Theta} \|y' - z\|.$$

For each resource i , this projection is given by a quadratic program with linear constraints, and can be solved efficiently using standard methods like a modification of the simplex (see Wolfe 1959) or barrier-type algorithms (see Bertsekas 1999, Chapter 4). Note that this projection typically involves a small number of variables and a small number of constraints.⁶

We finally note that in a commercial implementation, the simulations could be run on parallel processors, with each CPU generating its own sequence of demand and calculating the resulting sample path gradient. In this sense, the algorithm is highly parallelizable.

3.2. Convergence

Observe that in view of Theorem 1, our algorithm is unlikely to be globally convergent. However, it has at least robust local convergence properties. Recall that the gradient $\nabla_y R(y^{(k-1)}, \omega^{(k)})$ is a noisy representation of the gradient of $g(y^{(k-1)})$. Let the noise (error) in the gradient at iteration k be the vector $\xi^{(k)} \equiv \nabla_y R(y^{(k-1)}, \omega^{(k)}) - \nabla g(y^{(k-1)})$. From Lemma 2, we have that $E[\xi^{(k)} | y^{(0)}, \dots, y^{(k-1)}] = 0$.

Let the cumulative step sizes be defined as $s_k = \sum_{i=1}^{k-1} \rho^{(i)}$, and define a function $m(s)$ such that $m(s) = \max\{k: s_k \leq s\}$ for $s \geq 0$, and $m(s) = 0$ otherwise. Note

that from the choice of $\rho^{(k)}$ in (14), $m(s) < \infty$. Consider the following conditions:

ASSUMPTION A1. $\{\rho^{(k)}\}$ is a sequence of positive real numbers such that $\lim_{k \rightarrow \infty} \rho^{(k)} = 0$, $\sum_{k=1}^{\infty} \rho^{(k)} = +\infty$.

ASSUMPTION A2. Let the constraint set for the problem be defined by $\Theta = \{y: \theta_j(y) \leq 0, j = 1, \dots, s\}$. The set Θ is closed and bounded. The $\theta_j(\cdot)$, $j = 1, \dots, s$, are continuously differentiable. At each y that is on the boundary of Θ , the gradients of the active constraints are linearly independent.

ASSUMPTION A3. $\lim_{k \rightarrow \infty} \mathbb{P}(\sup_{k \leq l \leq m(s_k+s)} \|\sum_{i=k}^l \rho^{(i)} \xi^{(i)}\| > \epsilon) = 0$ for each $\epsilon > 0$ and $s > 0$.

ASSUMPTION A4. $g(\cdot)$ is a continuously differentiable real-valued function.

ASSUMPTION A5. $\rho^{(k)} E[\|\xi^{(k)}\|^2] \rightarrow 0$ as $k \rightarrow \infty$.

THEOREM 3. Let Θ^* be the set of Kuhn-Tucker points for the continuous problem (5)–(6). Then, the stochastic gradient algorithm described above verifies Assumptions A1–A5. Moreover, if Θ^* is a connected set, the sequence of points $y^{(k)} \rightarrow \Theta^*$ in probability as $k \rightarrow \infty$.

PROOF. We discuss here that Assumptions A1–A5 hold for our algorithm.

- A1 is satisfied by our choice of the step sizes $\rho^{(k)}$ in (14).

- A2 is satisfied by our constraint set (1) and our assumption that $x_i(1) > 0 \forall i$.

- For A3, note that for a fixed k , the sequence $\{\sum_{i=k}^l \rho^{(i)} \xi^{(i)}, l \geq k\}$ is a martingale. Because the Euclidean norm function is convex, we have that $\{\|\sum_{i=k}^l \rho^{(i)} \xi^{(i)}\|, l \geq k\}$ is a submartingale. Therefore,

$$\begin{aligned} & \mathbb{P}\left(\sup_{k \leq l \leq m(s_k+s)} \left\|\sum_{i=k}^l \rho^{(i)} \xi^{(i)}\right\| > \epsilon\right) \\ &= \mathbb{P}\left(\sup_{k \leq l \leq m(s_k+s)} \left(\sum_{i=k}^l \rho^{(i)} \xi^{(i)}\right)^2 > \epsilon^2\right) \\ &\leq \frac{E\left[\left(\sum_{i=k}^{m(s_k+s)} \rho^{(i)} \xi^{(i)}\right)^2\right]}{\epsilon^2} \\ &= \frac{\sum_{i=k}^{m(s_k+s)} [\rho^{(i)}]^2 E[[\xi^{(i)}]^2]}{\epsilon^2} \\ &\leq \frac{\sum_{i=k}^{\infty} [\rho^{(i)}]^2 E[[\xi^{(i)}]^2]}{\epsilon^2}, \end{aligned}$$

where the first inequality holds from Kolmogorov’s inequality for submartingales (e.g., see Ross 1996, Theorem 6.4.4), and the next equality holds because the noise terms are uncorrelated. Because of our choice of step sizes in (14) and bounded variance of the noise term (from Lemma 3), the last upper bound goes to zero as k goes to infinity.

- A4 holds by Theorem 2.
- A5 holds by the choice of $\rho^{(k)}$ and Lemma 3.

The convergence result follows from Theorem 6.3.1 in Kushner and Clark (1978). \square

A weaker convergence result holds when Θ^* is not connected. To do that, we first define an interpolation for the sequence $\{y^{(k)}\}$. We define the continuous function $y(s)$ by

$$y(s) = \begin{cases} y^{(k)} & \text{if } s = s_k, \\ \frac{s_{k+1} - s}{\rho^{(k)}} y^{(k)} + \frac{s - s_k}{\rho^{(k)}} y^{(k+1)} & \text{if } s \in (s_k, s_{k+1}). \end{cases}$$

Observe that $y(s)$ is just a linear interpolation of the values $y^{(k)}$ as a function of the cumulative step sizes s_k . Let $N_\epsilon(\Theta^*)$ denote the ϵ -neighborhood of the set Θ^* . Kushner and Clark (1978, Theorem 6.3.1) show that under A1–A5, if Θ^* is not connected, then for each $\delta > 0$ and $\epsilon > 0$, there exists a finite constant s_0 such that $s > s_0$ implies

$$\lim_{k \rightarrow \infty} \mathbb{P} \left(\frac{1}{2s} \int_{-s}^s \mathbf{I}\{y(s_k + v) \in \mathcal{R}^{m \times c} - N_\epsilon(\Theta^*)\} dv \geq \delta \right) \leq \delta.$$

Roughly, this result says that the “the average amount of time” the iterates $y^{(k)}$ lie more than ϵ away from a point in Θ^* (averaging over a sufficiently large but finite interval) becomes arbitrarily small as k increases. It is basically a convergence in probability of a “moving average” of $y^{(k)}$ rather than a convergence of $y^{(k)}$ itself.

Summarizing, the algorithm we proposed satisfies the conditions for the convergence to a Kuhn-Tucker point. When Θ^* is connected, we have convergence in probability. Even if Θ^* is not connected, we still have a guarantee of convergence of the average of iterates to a point arbitrarily close to Θ^* . We emphasize, though, that all these are only local convergence guarantees.

4. Numerical Experiments

In this section, we describe the results of some numerical tests of our algorithm. We first describe the implementation and experimental design details of our tests and then look at the results for three example networks.

4.1. Algorithm Implementation Details

For the virtual nesting scheme, we used a standard version of displacement adjusted virtual nesting (DAVN) as proposed in Williamson (1992). The method is based on solving a deterministic linear program based on the mean demands. The dual variables from this linear program are used to both compute displacement adjusted revenues and cluster products into virtual classes. The resulting single-leg models for each leg are solved using the EMSR-b (expected marginal seat revenue) heuristic of Belobaba (1992). Requests are processed through theft nesting. Our particular implementation of this method is described in detail in online Appendix A3. This overall scheme is

commonly used in practice and therefore serves as a good reference point for testing our algorithm.

As is common in practice, we also reoptimized the DAVN control periodically during the booking process. Specifically, we split the booking horizon into three periods and reoptimized the protection levels at the start of each period. We kept the original displacement adjusted revenues and mapping to virtual classes and only update the inputs to the single-leg problems (i.e., the current remaining capacity and demand-to-come statistics).

As for the gradient estimate, while we perturbed the remaining capacity in our original problem by a random noise term to ensure a.s. differentiability, our computations are mainly performed without introducing this perturbation. We indeed tested the perturbed version of the algorithm.⁷ However, note that the perturbation could be arbitrarily small—eventually smaller than the tolerance of any computer. In this case, one can avoid explicitly simulating this perturbation by noticing the following: If we focus on the a.s. differentiable control $u_j(x - \zeta, y, q)$ in (5)–(6), and consider its rate of change with respect to both y_{ic} and x_i when $\epsilon \rightarrow 0$, it corresponds to $(\partial^+/\partial y_{ic})u_j(x, y, q)$ and $(\partial^-/\partial x_i)u_j(x, y, q)$ (i.e., the right partial derivative with respect to y_{ic} and the left partial derivative with respect to x_i , respectively, because a negative perturbation in the remaining capacity is equivalent to a marginal increase in the protection levels and a marginal decrease in the available capacity). Observe that these directional partial derivatives always exist because the revenue function is piecewise linear. Moreover, they are defined by (10) and (11), albeit now with the upper limits described in conditions (ii) and (iii) replaced by nonstrict inequalities.

Given these observations, when ϵ is arbitrarily small, we can simply use the approximations

$$\begin{aligned} & \frac{\partial}{\partial y_{ic}} R_t(x(t), y, \omega) \\ & \approx \left(r_{j_i} - \sum_{k \in A_{j_i}} \frac{\partial}{\partial x_k} R_{t+1}(x(t+1), y, \omega) \right) \frac{\partial^+}{\partial y_{ic}} u_{j_i}(x(t), y, q_t) \\ & \quad + \frac{\partial}{\partial y_{ic}} R_{t+1}(x(t+1), y, \omega) \quad \forall i, c, t \end{aligned}$$

and

$$\begin{aligned} & \frac{\partial}{\partial x_i} R_t(x(t), y, \omega) \\ & \approx \left(r_{j_i} - \sum_{k \in A_{j_i}} \frac{\partial}{\partial x_k} R_{t+1}(x(t+1), y, \omega) \right) \frac{\partial^-}{\partial x_i} u_{j_i}(x(t), y, q_t) \\ & \quad + \frac{\partial}{\partial x_i} R_{t+1}(x(t+1), y, \omega) \quad \forall i, t \end{aligned}$$

This nonperturbed version produced exactly the same results as the perturbed version when applied over several examples when we set $\epsilon = 10^{-10}$; they showed negligibly different results when we set $\epsilon = 10^{-5}$. However, in terms of memory requirements and computational time, the

nonperturbed version is clearly more efficient.⁸ Because the nonperturbed version requires a simpler data structure, is computationally more efficient, and performed equally well in our numerical tests, this is the version that would likely be implemented in actual applications. Hence, we concentrated our tests on this version of the algorithm.

We have also tried computing the gradient based on a batch of sample paths instead of based on a single one, and then taking the average of them. The purpose is “smoothing” estimates as in the stochastic quasigradient method of Ermoliev (1988). Given a pool of simulated demand instances, our experiments suggest that it is more convenient to run a large number of iterations computing the gradient based on a single sample path, than to get an accurate estimate at each iteration through averaging the gradient computed over multiple sample paths, and running fewer iterations.

In terms of the computing platform, we implemented the stochastic gradient algorithm in C++ on a Pentium IV Workstation (CPU of 2.00 Ghz and RAM of 512 Mb) under Windows 2000.⁹

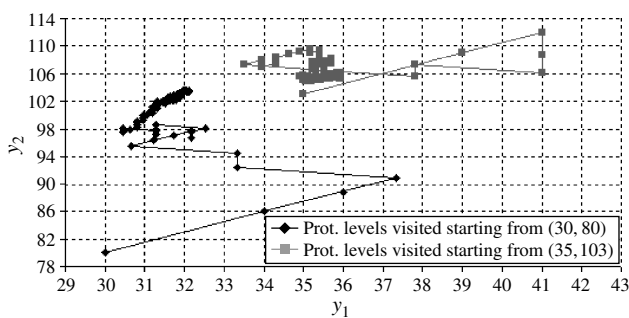
4.2. Examples

EXAMPLE 1. We start with a very small example to illustrate how the stochastic gradient algorithm evolves from an initial solution. Here, we have a single leg with capacity $x = 150$ and three different products (virtual classes). The revenues are $r = (200, 160, 100)$. Demands are assumed normal, with means $\mu = (40, 65, 70)$, standard deviations $\sigma_j = \sqrt{\mu_j}$, and distributions truncated between zero and $2\mu_j$. From these parameters, we calculate discrete distributions (a probability mass function is calculated from the normal c.d.f.). Customer arrival order is from low- to high-fare classes.

The same set of 200 sample paths is applied to two different initial sets of protection levels, and the evolution of the successive protection levels visited by the algorithm is shown in Figure 3. We chose a step length $\rho^{(k)} = 0.1/k$.

The first set of initial protection levels was $y^{(0)} = (30, 80)$. The output of the algorithm before rounding was $y^{(N)} = (32.11, 103.42)$. The sample average

Figure 3. Evolution of the protection levels visited by the stochastic algorithm in Example 1, when applying it over the same 200 sample paths.



revenue increased from 21,139 at $y^{(0)}$ to 22,209 at $\text{Round}(y^{(N)}) = (32, 103)$, corresponding to an expected revenue gap of 5.06% with a 95% confidence interval of $(-1.88\%, 12.00\%)$.

The second initial set was provided by EMSR-b: $y^{(0)} = (35, 103)$. The output of the algorithm before rounding was $y^{(N)} = (35.91, 105.94)$. The sample average revenue increased from 22,236 at $y^{(0)}$ to 22,276 at $\text{Round}(y^{(N)}) = (36, 106)$, corresponding to an expected revenue gap of 0.18% with a 95% confidence interval of $(-2.08\%, 2.44\%)$.

Figure 3 illustrates the behavior of the stochastic algorithm from these two starting points. The pattern of progress observed here is typical of the algorithm behavior with decreasing jump sizes and oscillating behavior. (See Gaivoronski 2005 for other numerical examples exhibiting similar convergence patterns for stochastic quasigradient methods.) Despite the small number of instances generated, note that in both cases, the algorithm brought the initial protection levels close to the near-optimal ones given by EMSR-b (widely used in the airline practice), yet still showing some improvement over them.

However, one must be careful not to interpret Figure 3 as representing convergence to a (locally optimal) solution per se. First, the decreasing step sizes mean that the algorithm’s progress naturally “stalls” after a large number of iterations simply because the step sizes become very small. Second, we terminated the algorithm after a fixed number of iterations. At best, therefore, one can claim the algorithm moves the protection levels to an “improved” set of values. Convergence to optimal values would likely take many more iterations. Still, as a practical matter, achieving an improvement in the protection levels is the main aim of the algorithm and by this measure it appears to do well.

EXAMPLE 2. This example is based on the five-airport network shown in Figure 4 and data from Williamson (1992). There are 10 round-trip itineraries. Each itinerary is segmented into four different classes: Y, M, B, and Q (see Table 3). Each product is a combination of one-way itinerary and class (e.g., product 1 is the one-way itinerary ATL-BOS for class Y, which has revenue $r_1 = 310$ and mean demand $\mu_1 = 12$; product 2 corresponds to ATL-BOS, class M, with revenue $r_2 = 290$ and mean demand $\mu_2 = 9$; product 5 is BOS-ATL, class Y, with associated $r_5 = 310$ and

Figure 4. Five-airport network for Example 2.

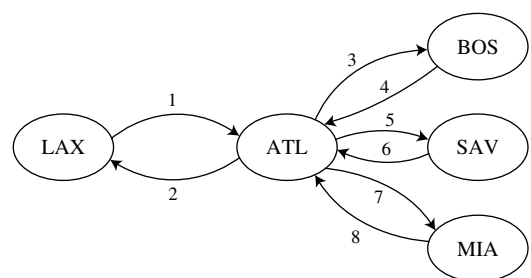


Table 3. Description of the network in Figure 4.

O-D market	Revenue				Mean demand			
	Y	M	B	Q	Y	M	B	Q
ATLBOS/BOSATL	310	290	95	69	12	9	11	17
ATLLAX/LAXATL	455	391	142	122	8	4	11	27
ATLMIA/MIAATL	280	209	94	59	20	9	7	14
ATLSAV/SAVATL	159	140	64	49	25	7	5	13
BOSLAX/LAXBOS	575	380	159	139	9	7	11	24
BOSMIA/MIABOS	403	314	124	89	11	5	14	20
BOSSAV/SAVBOS	319	250	109	69	5	7	11	27
LAXMIA/MIALAX	477	239	139	119	17	11	7	14
LAXSAV/SAVLAX	502	450	154	134	5	7	7	32
MIASAV/SAVMIA	226	168	84	59	13	4	7	25

$\mu_5 = 12$, etc). This gives eight legs in the network and 80 products.

We assume that demand arrives from lower to higher in revenue order: First, requests for products $j = 4, 8, 12, \dots, 80$ arrive (class Q); followed by requests for products $j = 3, 7, 11, \dots, 79$ (class B); then by products $2, 6, 10, \dots, 78$ (class M), and finally by products $1, 5, 9, \dots, 77$ (class Y).

Demand for product j was assumed to be normally distributed with mean $E[D_j] = \mu_j$ (provided in Table 3, as explained above) and standard deviation $\sigma_j = \sqrt{\mu_j}$. This distribution was then truncated between zero and $2\mu_j$. From the resulting continuous c.d.f., a probability mass function was then calculated. That is, demand was treated as a discrete random variable such that for a given integer value a , the probability that demand was equal to a was taken as $\int_{a-1/2}^{a+1/2} f(x) dx$, where $f(x)$ is the continuous density function. We assumed that each customer requests exactly one seat ($q_t = 1$ for all t). The maximum number of protection levels per leg was set at $\bar{c} = 9$ (i.e., there are 10 virtual classes).

For the stochastic algorithm, we used a step size of $\rho^{(k)} = 0.1/k$. Kleywegt and Shapiro (2001) point out that this sort of method is very sensitive to the choice of step sizes: Small step sizes result in very slow progress toward the optimum, while large step sizes make the iterations zigzag. We experimented with $\rho^{(k)} = a/k$, with $a = 0.01, 0.05, 0.1, 0.5$, and 1 . The best results were obtained with $a = 0.1$. DAVN was used to define the indexing and find the initial set of protection levels as described in online Appendix A3. When running the stochastic gradient method, we generated 5,000 instances (sample paths) taking as a starting point the

original outcome of DAVN.¹⁰ Each sample path consisted of 1,030 requests on average. As mentioned, we then split the booking horizon evenly into three periods and reoptimized the protection levels for both methods at the start of each period.

Two measures of congestion were computed for reference: the *demand factor* is a measure of the level of demand relative to capacity. It is computed as the average of the *leg demand factors*, where

$$\text{demand factor for leg } i = \frac{\sum_{j=1}^n E[D_j] \mathbf{I}\{i \in A_j\}}{x_i}. \quad (15)$$

Demand factors close to or above one indicates scarcity of resources and the need of some capacity control to optimize revenues. The second congestion measure is the *load factor*, an indicator of capacity utilization. It is also computed as the average of the *leg load factors*:

$$\text{load factor for leg } i = \frac{\text{average number of seats sold on leg } i}{x_i}. \quad (16)$$

Note that the load factor depends on the capacity control policy, while the demand factor is independent of the capacity control policy. We varied the demand factors and load factors by scaling the capacity on each leg of the network, leaving demand unchanged.

Table 4 shows average revenues and average load factors for different capacity levels. We also list the 95% confidence intervals for the revenue gaps. It took approximately 35 seconds to compute the new set of protection levels starting from the original output of DAVN. These new protection levels produced significant increases in revenues—the expected gaps are between 1.74% and 5.15%—and also slightly higher load factors, which suggests that not only were more requests accepted but also the mix of accepted demand improved.

To illustrate the changes in protection levels, Tables 5 and 6 show the protection levels before and after execution of the SA algorithm, computed at the start of the first booking period for the case $x_i = 180$ for all legs i . For this example, there were at most eight virtual classes per leg. It is interesting to observe the structure of the protection levels produced by the stochastic gradient algorithm. Some levels are increased, while others are decreased. Indeed, some virtual classes are in fact collapsed (e.g., virtual classes 4

Table 4. Revenues for different capacity levels for Example 2.

Capacity per leg	Demand factor	Perf. for reopt. DAVN		Performance for SA				
		Revenue	Load	Revenue	Load	E[% Gap]	% Gap (95% CI)	CPU sec.
160	1.25	157,640	0.88	165,758	0.91	5.15	(3.09, 7.21)	32.86
180	1.11	165,270	0.89	173,050	0.90	4.71	(2.67, 6.74)	36.72
200	1.00	174,707	0.90	181,424	0.91	3.84	(2.03, 5.65)	36.84
220	0.91	183,668	0.88	186,865	0.89	1.74	(−0.20, 3.68)	34.92

Table 5. Initial protection levels $y^{(0)}$ for Example 2, when $x_i = 180$, for the first period in the booking horizon.

Resource	Protection levels y_{ic}						
	1	2	3	4	5	6	7
1	6	34	42	49	61	75	88
2	6	34	42	49	61	75	88
3	7	18	30	59	73	103	—
4	7	18	30	59	73	103	—
5	3	11	18	38	63	81	96
6	3	11	18	38	63	81	96
7	25	52	62	79	98	107	141
8	25	52	62	79	98	107	141

and 5 are merged on leg 2 because $y_{23} = y_{24}$; virtual classes 3 and 4 are merged on legs 7 and 8 because $y_{72} = y_{73}$ and $y_{82} = y_{83}$). Virtual class 1 disappeared from the first four legs, meaning that there is no specific capacity reserved for it.

EXAMPLE 3. This example is a real-world data set based on a subnetwork from a major U.S. commercial airline. The network has 1,844 products (origin-destination-fare-classes) and 62 legs. Each stream of demand consists of the order of 7,700 bookings. Again, we assumed that each customer requests a single seat ($q_t = 1$ for all t). Demand for each product was assumed to be a truncated normal random variable, or a truncated Poisson random variable when the mean was less than five. The mean demand per product varied from less than one to 86. We allowed a maximum of 15 virtual classes per leg, and computed the indexing and initial protection levels using DAVN as described in online Appendix A3. We again split the horizon into three equal-sized periods and reoptimized the protection levels at the start of each period. A thousand streams of demand are generated as input to our SA algorithm. We used the step size $\rho^{(k)} = 0.1/k$, as in Example 2.

Table 7 shows average revenues, load factors, and revenue gaps for different demand factors. The revenue gains

Table 6. Improved protection levels $y^{(N)}$ for Example 2, when $x_i = 180$, for the first period in the booking horizon.

Resource	Protection levels y_{ic}						
	1	2	3	4	5	6	7
1	0	34	41	42	45	75	81
2	0	34	41	41	44	74	81
3	0	0	9	47	71	105	—
4	0	3	6	45	73	104	—
5	3	9	10	10	59	81	87
6	3	11	13	13	61	81	90
7	25	42	42	79	98	107	165
8	21	39	39	77	98	107	159

are less dramatic than in Example 2 but are still significant, ranging from 0.56%–1.32%.

The stochastic gradient algorithm computes the first set of protection levels in around one minute, which is a remarkable time given the size of the problem. (The time for computing the two reoptimized sets of protection levels is even shorter because they are based on the residual demand-to-come, which results in smaller sample paths.) Overall, the example suggests that the approach has good practical potential.

5. Conclusions

In this paper, we have proposed a model and algorithm that improves on the simulation-based approach developed by Bertsimas and de Boer (2005). The method has the advantage of being simple to implement, is computationally efficient, and provably convergent.

We believe the model and solution approach are appealing for several reasons. First, the method improves protection levels based on an accurate estimate of true network effects, without relying on approximations or decompositions. Second, because it is simulation based, the demand processes can be completely general. For example, it could include correlations among itineraries or correlations over time. Third, it is easy to implement, requiring a recursive iteration that is not much more complex than simulating the acceptance decisions themselves. Fourth, due to its continuous nature, our model overcomes the computational and theoretical difficulties of working with finite-difference estimates and a discrete optimization problem as in the work of Bertsimas and de Boer (2005). Finally, it shows promising results in our computational tests and seems reasonably efficient even for some relatively large, real-world networks.

As for future research, variations of the basic computational method would be worth investigating. Ours is a straightforward implementation of a stochastic gradient method, and as such primarily serves to illustrate the value of gradient estimators and the potential performance improvements they yield. However, it may be desirable to develop a sample average approximation version of the algorithm that exploits more advanced methods in nonsmooth optimization (e.g., bundle methods). Variance reduction methods would be worth exploring as well.

Also, it would be worthwhile to try to apply a similar approach to bid-price controls. The overall approach can also be applied to cases where customers exhibit more complex behavior. For example, where each customer has an ordered preference for products to buy and if their first choice is not available, they buy the second one if it is available, and so on. This allows modeling of path preference and buy-up and buy-down behavior which is not taken into account by traditional methods (DAVN, EMSR-b, bid prices, etc). This extension is the topic of a recent paper (van Ryzin and Vulcano 2008).

Table 7. Revenues for different demand factors for Example 3.

Demand factor	Perf. for reopt. DAVN		Performance for reoptimized SA				
	Revenue	Load	Revenue	Load	E[% Gap]	% Gap (95% CI)	CPU sec.
1.28	2,318,515	0.88	2,331,538	0.92	0.56	(−0.41, 1.54)	68.80
1.10	2,496,684	0.86	2,525,456	0.89	0.65	(0.46, 1.84)	60.53
0.96	2,648,337	0.83	2,683,401	0.85	1.32	(0.86, 1.79)	63.51
0.86	2,781,869	0.81	2,816,914	0.83	1.26	(0.92, 1.60)	57.09
0.77	2,906,919	0.78	2,928,350	0.79	0.73	(0.36, 1.11)	55.20

6. Electronic Companion

An electronic companion to this paper is available as part of the online version that can be found at <http://or.journal.informs.org/>.

Acknowledgments

The authors thank Sanne de Boer, Paul Glasserman, and Anton Kleywegt for helpful discussions on earlier drafts of this work. They also thank Vladimir Norkin for comments on an earlier version of this paper. Andy Boyd of PROS Revenue Management and Shankar Sivaramakrishnan provided some of the data sets used in the numerical experiments. Finally, the authors thank the Deming Center at the Columbia Business School for having partially supported this research.

Endnotes

1. Again, while ζ_t is a function of ω , we do not explicitly express this dependence to simplify the notation.
2. This complexity relies on the reasonable assumption that there is a small constant upper bound for the number of resources used by each product. For example, in the airline case we could reasonably take a value of four, which corresponds to having itineraries with at most three stopovers. Then, when a request hits a protection level, the number of components to update in the gradient vector is also small (e.g., at most $4\bar{c}$).
3. We implemented both the stochastic gradient and finite-difference algorithms in Microsoft Visual C++ 6.0 on a Pentium IV Workstation (CPU of 2.00 Ghz and RAM of 512 Mb) under Windows 2000.
4. A proxy for measuring the network congestion is the demand factor. Its formal definition and further description of the network examples are provided in §4, Example 2 (for the first four cases in Table 1) and Example 3 (for the last five cases).
5. For simplicity, we do not perturb capacity in this sample path example; moreover, because these perturbations can be taken arbitrarily small, including them does not change the example.
6. Gu (2002) reported that for small models, variations of both primal and dual simplex for quadratic programming outperforms the barrier-type algorithms in terms of computational time.

7. In the perturbed version, we have also simulated continuous quantities for the different requests, assuming that they were $\text{Unif}(-\epsilon + q, q + \epsilon)$ for a discrete quantity q in the sample path ω .

8. Regarding the memory requirements of the perturbed algorithm, one has to keep track of all the perturbations for every request t in the sample path. As for computational time, in Example 2, it took roughly 35 seconds to solve for the nonperturbed version (see Table 4) and 47 seconds for the perturbed version.

9. We have worked with Microsoft Visual C++ 6.0, building a Win32 console application. We have linked our code with a LINDO application programming interface (Lindo Systems, Inc.) to make the projection in Step 2(d). This routine uses a barrier algorithm to solve the quadratic program.

10. Both the implementation of DAVN and the sample path simulation were computed offline with MATLAB from MathWorks Inc.

References

- Belobaba, P. 1987. Air travel demand and airline seat inventory management. Ph.D thesis, Flight Transportation Laboratory, Massachusetts Institute of Technology, Cambridge, MA.
- Belobaba, P. 1989. Application of a probabilistic decision model to airline seat inventory control. *Oper. Res.* **37** 183–197.
- Belobaba, P. 1992. Optimal vs. heuristic methods for nested seat allocation. *Proc. AGIFORS Reservations and Yield Management Study Group*, Brussels, Belgium, 28–53.
- Belobaba, P. 2001. Revenue and competitive impact of O-D control: Summary of PODS results. First Annual INFORMS Revenue Management Section Meeting, Columbia University, New York.
- Belobaba P., S. Lee. 2000. PODS update: Large network O-D control results. AGIFORS Revenue Management Study Group Meeting, New York.
- Benveniste A., M. Métivier, P. Priouret. 1990. *Adaptive Algorithms and Stochastic Approximations*. Springer-Verlag, Berlin.
- Bertsekas, D. 1999. *Nonlinear Programming*, 2nd ed. Athena Scientific, Nashua, NH.
- Bertsimas, D., S. de Boer. 2005. Simulation-based booking-limits for airline revenue management. *Oper. Res.* **53** 90–106.
- Billingsley, P. 1995. *Probability and Measure*. John Wiley & Sons, New York.
- Brumelle, S., J. McGill. 1993. Airline seat allocation with multiple nested fare classes. *Oper. Res.* **41** 127–137.
- Curry, R. 1990. Optimal airline seat allocation with fare classes nested by origins and destinations. *Transportation Sci.* **24** 193–204.
- de Boer, S. 2003. Personal communication.
- Dror, M., P. Trudeau, S. Ladany. 1988. Network models for seat allocation on flights. *Transportation Res.* **22B** 239–250.

- Ermoliev, Y. 1988. Stochastic quasigradient for methods. Y. Ermoliev, R. J.-B. Wets, eds. *Numerical Techniques Stochastic Optimization*, Chapter 6. Springer-Verlag, Berlin, 143–185.
- Gaivoronski, A. 2005. SQG: Software for solving stochastic programming problems with stochastic quasigradient methods. S. Wallace, W. Ziemba, eds. *Applications of Stochastic Programming*. MPS/SIAM Series in Optimization, SIAM, Philadelphia, 37–60.
- Glasserman, P. 1994. Perturbation analysis of production networks. D. D. Yao, ed. *Stochastic Modeling and Analysis of Manufacturing Systems*, Chapter 6. Springer, New York, 233–280.
- Glover, F., R. Glover, J. Lorenzo, C. McMillan. 1982. The passenger mix problem in the scheduled airlines. *Interfaces* **12** 72–79.
- Gu, Z. 2002. Quadratic programming and mixed integer quadratic programming: Algorithms and implementation. Presented at Second Columbia Optimization Day. Columbia University, New York.
- Karaesmen, I., G. J. van Ryzin. 2004. Overbooking with substitutable inventory classes. *Oper. Res.* **52** 83–104.
- Kleywegt, A. J., A. Shapiro. 2001. Stochastic optimization. G. Salvendy, ed. *The Handbook of Industrial Engineering*, 3rd ed. John Wiley & Sons, New York, 2625–2650.
- Kushner, H. J., D. S. Clark. 1978. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, Berlin.
- Kushner, H. J., G. Yin. 2003. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, New York.
- L'Ecuyer, P., P. Glynn. 1994. Stochastic optimization by simulation: Convergence proofs for the GI/G/1 queue in steady state. *Management Sci.* **40** 1562–1578.
- L'Ecuyer, P., N. Giroux, P. Glynn. 1994. Stochastic optimization by simulation: Numerical experiments with the M/M/1 queue in steady state. *Management Sci.* **40** 1245–1261.
- Lee, T., M. Hersh. 1993. A model for dynamic airline seat inventory control with multiple seat bookings. *Transportation Sci.* **27** 252–265.
- Lemaréchal, C. 1989. Nondifferentiable optimization. G. Nemhauser, A. Rinnooy Kan, M. Todd, eds. *Handbooks in OR & MS*, Vol. 1, Chapter VII. Elsevier Science Publishers, 529–572.
- Littlewood, K. 1972. Forecasting and control of passengers bookings. *12th AGIFORS Sympos. Proc.*, Nathanya, Israel, 95–128.
- Mahajan, S., G. van Ryzin. 2001. Stocking retail assortments under dynamic consumer substitution. *Oper. Res.* **49** 334–351.
- Pflug, G. 1988. Step size rules, stopping times and their implementation in stochastic quasigradient methods. Y. Ermoliev, R. J.-B. Wets, eds. *Numerical Techniques for Stochastic Optimization*, Chapter 17. Springer-Verlag, Berlin, 353–372.
- Plambeck, E. L., B. Fu, S. M. Robinson, R. Suri. 1996. Sample-path optimization of convex stochastic performance functions. *Math. Programming* **75** 137–176.
- Robbins, H., S. Monro. 1951. On a stochastic approximation method. *Ann. Math. Statist.* **22** 400–407.
- Ross, S. M. 1996. *Stochastic Processes*, 2nd ed. John Wiley & Sons, New York.
- Shapiro, A. 2000. Stochastic programming by Monte Carlo simulation methods. *Stochastic Programming E-Prints Series*, article 2000–2003.
- Smith, B., C. Penn. 1988. Analysis of alternative origin-destination control strategies. *AGIFORS Sympos. Proc.*, Vol. 28. New Seabury, MA, 123–144.
- Smith, B., J. Leimkuhler, R. Darrow. 1992. Yield management at American Airlines. *Interfaces* **22** 8–31.
- Talluri, K., G. J. van Ryzin. 2004. *The Theory and Practice of Revenue Management*. Kluwer Academic Publishers, New York.
- van Ryzin, G., G. Vulcano. 2008. Computing virtual nesting controls for network revenue management under customer choice behavior. *Manufacturing Service Oper. Management* **10**(3) 448–467.
- Williamson, E. 1992. Airline network seat inventory control: Methodologies and revenue impacts. Ph.D. thesis, Flight Transportation Laboratory, Massachusetts Institute of Technology, Cambridge, MA.
- Wolfe, P. 1959. The simplex method for quadratic programming. *Econometrics* **37** 382–398.