INFORMS Journal on Computing
Vol. 25, No. 3, Summer 2013, pp. 488–501
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.1120.0516

Scenario Trees and Policy Selection for Multistage Stochastic Programming Using Machine Learning

Boris Defourny
Department of Operations Research and Financial Engineering, Princeton University, Princeton, New Jersey 08544,
defourny@princeton.edu

Damien Ernst, Louis Wehenkel
Department of Electrical Engineering and Computer Science, University of Liège, 4000 Liège, Belgium
{dernst@ulg.ac.be, L.Wehenkel@ulg.ac.be}

In the context of multistage stochastic optimization problems, we propose a hybrid strategy for generalizing to nonlinear decision rules, using machine learning, a finite data set of constrained vector-valued recourse decisions optimized using scenario-tree techniques from multistage stochastic programming. The decision rules are based on a statistical model inferred from a given scenario-tree solution and are selected by out-of-sample simulation given the true problem. Because the learned rules depend on the given scenario tree, we repeat the procedure for a large number of randomly generated scenario trees and then select the best solution (policy) found for the true problem. The scheme leads to an ex post selection of the scenario tree itself. Numerical tests evaluate the dependence of the approach on the machine learning aspects and show cases where one can obtain near-optimal solutions, starting with a "weak" scenario-tree generator that randomizes the branching structure of the trees.

Key words: stochastic programming; out-of-sample validation; artificial intelligence
History: Accepted by David Woodruff, Area Editor for Heuristic Search and Learning; received December 2011; revised April 2012; accepted April 2012. Published online in *Articles in Advance* August 14, 2012.

1. Introduction

Stochastic optimization using scenario trees has proven to be a powerful algorithmic strategy but has suffered from the rapid growth in the size of scenario trees as the number of stages grows (Birge and Louveaux 1997, Shapiro et al. 2009). A number of authors have undertaken research to limit the size of the scenario tree, but problem size still grows exponentially with the number of stages (Frauendorfer 1996, Dupacova et al. 2000, Høyland and Wallace 2001, Pennanen 2009, Heitsch and Römisch 2009). As a result, most authors either severely limit the number of decision stages or sharply limit the number of scenarios per stage (Birge 1997, Wallace and Ziemba 2005, Dempster et al. 2008, Kallrath et al. 2009). Such approximations make it possible to optimize first-stage decisions with a stochastic look-ahead but without tight guarantees on the value of the computed decisions for the true multistage problem (as a matter of fact, bounding techniques also tend to break down on problems with many stages).

Some authors have proposed to assess the quality of scenario-tree based methods by out-of-sample validation (Kouwenberg 2001, Chiralaksanakul 2003, Hilli and Pennanen 2008). The validation scheme consists of solving the multistage program posed on a scenario tree spanning the planning horizon T, implementing the decision relative to time step 1, sampling the realization of the stochastic process at time 1, updating the conditional distributions of the stochastic process from time 2 to time T, rebuilding a scenario tree spanning time periods 2 to T, solving the new multistage program over the remaining horizon (with previously implemented decisions fixed to their value), and continuing this process until the last decision at time T has been found. The resulting sequence of decisions is then valued according to the true objective function. Averaging the result of this procedure repeated over many independent scenarios drawn according to the true distributions of the problem produces an unbiased estimate of the expected value of the solution for the true problem. Unfortunately, such simulations are very demanding computationally. Moreover, the variance of the empirical estimate is likely to be larger for problems with many stages, calling for even more simulations in that case. As a result, running times restrict the use of this technique to relatively simple optimization problems and simple scenario-tree updating schemes.

In this paper, we propose a hybrid approach that combines scenario trees with the estimation of statistical models for returning a decision given a state. One could call these statistical models *policy function* *approximations* (PFAs). We solve an optimization problem on a scenario tree to obtain optimal decisions (given the tree), and then we use the decisions at all stages in the tree to fit the policy function approximations using methods based on supervised learning (Hastie et al. 2009). We repeat this exercise for different samples of scenario trees, producing a family of policy function approximations. Each policy function approximation is then tested on a fresh set of samples to determine the best policy function approximation.

Machine learning methods have often been applied to *stochastic optimization*, primarily in the context of approximating a value function (Berstsekas and Tsitsiklis 1996, Sutton and Barto 1998, Bertsekas 2005, Busoniu et al. 2010, Szepesvári 2010, Powell 2011). The statistical estimation of policies has also been widely studied in the reinforcement learning community, often using the term "actor-critic" methods (Sutton and Barto 1998, Peters and Schaal 2008). Such methods are popular in computer science for discrete action spaces and in control theory for low-dimensional but unconstrained control problems. Our method is designed for higher dimensional constrained optimization problems.

In general, the constraints of the problem induce a class of admissible policies that is not ideal for the purpose of doing regression. Typically, with vector-valued decisions it may be best to use a smooth class of functions that is less likely to overfit the decisions from the scenario tree, although ultimately the recourse decisions must satisfy the constraints exactly. These aspects are reconciled in our work by adding a corrective step to the PFA that consists of solving a constrained optimization problem for finding a solution that minimizes the deviation from the decision predicted by a less constrained statistical model.

We note that some authors have also proposed to derive a policy from a scenario tree by applying to a new scenario the decision optimized for the closest scenario in the tree (Thénié and Vial 2008, Küchler and Vigerske 2010); their strategy could be viewed as a form of apprenticeship learning by nearest neighbor regression (Abbeel and Ng 2004, Syed et al. 2008, Coates et al. 2008). However, the use of machine learning along with a valid model selection procedure is quite new in the context of *stochastic programming*, whereas the need for methods able to discover automatically good decision rules had been recognized as an important research direction for addressing complex multistage stochastic programming problems (Mulvey and Kim 2011) and for bounding approximation errors (Shapiro 2003).

The machine learning approach makes it possible to quickly perform out-of-sample evaluations of the policy function approximations created using each scenario tree. The result is that it is now practical to compute safe statistical guarantees when using a scenario-tree approximation scheme.

Building on this ability, we propose to revisit the fundamental problem of generating the scenario tree from which a first-stage decision to be implemented is optimized. We are particularly interested in working with trees that have a sparse branching structure for representing uncertainty over a long planning horizon using a limited number of scenarios. Optimization over long planning horizons is especially relevant for exploiting a resource in limited quantity such as water in a reservoir, or an electricity swing option, when the price of the resource is stochastic. Small trees are also a pragmatic choice when the number of scenarios is very limited by the complexity and dimension of the problem, for instance in stochastic unit commitment problems for electricity generation scheduling.

In this paper, we consider a randomized algorithm for generating small scenario trees over long horizons that uses a branching process for generating the branching structure. We illustrate the solution approach on a set of problems over a long planning horizon. A small fraction of the PFAs learned from the random scenario trees turn out to perform very well in out-of-sample simulations. We need not ask more from the scenario-tree generation algorithm, in sharp contrast with solutions approaches based on the optimization of a single scenario tree.

Our approach to scenario-tree generation can be seen as an extension of the stochastic approximation method (SAA; Shapiro et al. 2009), where in addition to drawing scenarios randomly, the branching structure of the tree is also drawn randomly. However, because the detection of good structures is left to the out-of-sample validation, the way of thinking about the scenario trees is radically changed: in our solution approach, the best-case behavior of the scenario-tree generation algorithm can be exploited. Our approach could be contrasted to other solution schemes based on multiple trees (possibly each reduced to one scenario) used inside averaging or aggregation procedures (Mak et al. 1999, Nesterov and Vial 2008), perhaps most notably in Rockafellar and Wets (1991), which inspired the title of this paper.

Our paper makes the following contributions. We introduce the hybrid policy structure along with a model selection procedure for selecting a best policy function approximation, given a scenario-tree solution and the specification of the probability distributions and constraints of the true problem. This idea was originally presented in Defourny et al. (2009), where complexity estimates were given (but without working algorithms). We identify statistical models amenable to fast simulation in the context of convex multistage stochastic programming problems so as to be able to quickly generate and simulate feasible decisions on a large test sample. We conduct numerical tests on a multistage stochastic problem to assess the sensitivity of the approach to various choices and its benefit in terms of running time for obtaining a solution with a statistical performance guarantee. We propose a novel way to view the problem of constructing the scenario trees that has the potential to scale better with the number of decision stages without imposing stringent conditions on the problem structure, the probability distributions, or a class of decision rules. We report successful numerical experiments obtained with this technique.

The remainder of this paper is organized as follows. Section 2 explains the solution approach. Section 3 formalizes the description and gives algorithmic procedures for its implementation. Section 4 investigates the method numerically on a test problem. Section 5 describes the proposed application of machine learning techniques to scenario-tree selection and reports numerical results; §6 concludes.

2. Principle

Let us explain the principle of our solution approach on a stylized example. Consider decision stages numbered from $t=1$ to $t=T$ with T large, say $T=50$. Assume that the decision at stage t is given by a function $x_t(\cdot)$ of the history $\xi_{[t]} = (\xi_1, \xi_2, \ldots, \xi_t)$ of a random process $\xi_1, \ldots, \xi_T$, where by convention ξ_1 is deterministic. Suppose for concreteness that ξ_t for $t \geq 2$ follows the standard normal distribution and that $x_t(\cdot)$ must be valued in the interval $[0,1]$. We can write the multistage problem as $\mathcal{P}$: $\min \mathbb{E} f(x_1, \xi_2, x_2(\xi_{[2]}), \ldots, \xi_T, x_T(\xi_{[T]}))$, assuming that one observes ξ_t and then takes decision $x_t(\xi_{[t]})$; the minimization is over x_1 and the functions $x_t(\cdot)$. By convention $f(\cdot) = +\infty$ if $x_t(\xi_{[t]}) \notin [0,1]$ for any t. Let us denote by $\bar{v}$ the optimal value of the problem, assumed to be finite.

Assume first that a scenario tree $\mathcal{T}$ is given to us. Each node j of the tree represents a particular information state $h_j = (\xi_1, \ldots, \xi_{t_j})$, where t_j is the stage index determined by the depth of node j. Transition probabilities are associated to the arcs between successor nodes. Each leaf node k of the tree determines, by its path from the root node, a particular scenario $\xi^k = (\xi_1, \xi_2^k, \ldots, \xi_T^k)$. The probability of the scenario, written p^k, is obtained by multiplying the transition probabilities of the arcs of the path. Thus, the probability of reaching an information state h_j is the sum of the probabilities p^k of the scenarios passing through that node. Note that most nodes can have only one successor node because otherwise the number of scenarios would be an astronomical number on this problem with so many stages.

On the scenario tree, we can formulate a math program where optimization variables are associated to the nodes of the tree. For each scenario k, we associate optimization variables $x_1^k, \ldots, x_T^k$ to the nodes on the path from the root to the leaf k, and we enforce identity among the optimization variables that share a common node in the tree (nonanticipativity constraints). We solve $\mathcal{P}(\mathcal{T})$: $\min \sum_k p^k f(x_1^k, \xi_2^k, x_2^k, \ldots, \xi_T^k, x_T^k)$ subject to the nonanticipativity constraints. Let $v(\mathcal{T})$ and $x_1^*(\mathcal{T})$ denote its optimal value and optimal first-stage decision given the tree.

At this stage, the regret of implementing $x_1^*(\mathcal{T})$, that is,

$$\min_{x_1, x_2(\cdot), \ldots, x_T(\cdot)} \quad \mathbb{E} f(x_1, \xi_2, x_2(\xi_{[2]}), \ldots, \xi_T, x_T(\xi_{[T]})) - \bar{v}$$
$$\text{subject to} \quad x_1 = x_1^*(\mathcal{T}),$$

is unknown: the tree represents the distribution of ξ_t given $\xi_{[t-1]}$ by a single realization at most of the nodes, so the regret depends on the value of the stochastic solution for the subproblem at each node.

Still, an optimal solution to the problem formulated on the scenario tree provides examples of input-output pairs $(\xi_{[t]}^k, x_t^k)$. By machine learning, we can infer (learn) a statistical model for each mapping $x_t(\cdot)$. In §3, the idea is generalized to vector-valued decisions and convex feasibility sets by building one model per coordinate and restoring the structure of the vector by solving a small math program. For simplicity here, we illustrate the idea in the scalar case and with parametric regression. By a set of points in the input-output space $\mathbb{R}^t \times [0,1]$, we can fit a constant-valued function, a linear function, a quadratic function, and so forth (Figure 1); feasibility can be ensured by squashing back the output of $x_t(\cdot)$ to $[0,1]$. We can build statistical models $\hat{x}_t(\cdot)$ for $x_t(\cdot)$ and define a policy function approximation $\pi = (\pi_1, \ldots, \pi_T)$ for making decisions from stage 1 to T, where π_1 is set to $x_1^*(\mathcal{T})$, and π_t has values $\pi_t(\xi_{[t]}) = \text{squash}_{[0,1]} \hat{x}_t(\xi_{[t]})$. Among a set of policies π^ν, where ν indexes the model, we cannot know in advance the model that works best for the true problem. However, we can sample N new independent scenarios $(\xi_1, \xi_2^n, \ldots, \xi_T^n)$ according to the true distributions (in this example, standard normal distributions) and guarantee, if N is sufficiently large, that

$$\bar{v} \leq \min_\nu \left\{ \frac{1}{N} \sum_{n=1}^N f(\pi_1, \xi_2^n, \pi_2^\nu(\xi_1, \xi_2^n), \ldots, \xi_T^n, \pi_T^\nu(\xi_1, \xi_2^n, \ldots, \xi_T^n)) + \mathcal{O}(N^{-1/2}) \right\},$$

where the right-hand side is simple to evaluate because of the nature of the models π_t^ν (the $\mathcal{O}(N^{-1/2})$ term is a standard error term detailed in §3.7).

At this stage, we have not modified the first-stage solution $x_1^*(\mathcal{T})$ given the tree $\mathcal{T}$. The next step, which allows us to convert the tree evaluation method to a search method for x_1, is to have the whole process repeated for a large number of different trees, where each tree has potentially a different first-stage decision. Given an algorithm $\mathcal{A}$ that maps a vector of parameters $\theta_{\mathcal{A}}^m$ to a scenario tree $\mathcal{T}^m$, we can generate M trees by selecting or sampling $\theta_{\mathcal{A}}^m$ and from each tree $\mathcal{T}^m$ learn models $\pi^{m,\nu}$ from the solution to $\mathcal{P}(\mathcal{T}^m)$. Using a large sample of N scenarios, we can guarantee, if N is sufficiently large, that

$$\bar{v} \le \min_{m,\nu} \left\{ \frac{1}{N} \sum_{n=1}^{N} f(\pi_1^m, \xi_2^n, \pi_2^{m,\nu}(\xi_1, \xi_2^n), \dots, \xi_T^n, \pi_T^{m,\nu}(\xi_1, \xi_2^n, \dots, \xi_T^n)) + \mathcal{O}(N^{-1/2}) \right\}.$$

We can then implement the decision π_1^m of the tree that attains the minimum or even use the best model $\pi^{m,\nu}$ over the whole horizon. Note that an unbiased estimate of the expected value of the selected model $\pi^{m,\nu}$ on the true problem is obtained by simulating the model again on a new independent test sample of scenarios.

For trees over a large planning horizon, we propose in §5.2 to consider algorithms $\mathcal{A}$ capable of generating different branching structures given a target number of scenarios so as to keep the complexity of $\mathcal{P}(\mathcal{T}^m)$ under control while obtaining diversity in the trees.

In the remainder of the paper, we formalize this approach using nonparametric statistical models and develop methods for dealing with vector-valued decisions and feasibility constraints. A first set of numerical experiments compares the bounds obtained with the machine learning approach to the ideal bound computed by solving a multistage program at each stage for each out-of-sample scenario (on a problem over four stages). A second set of numerical experiments evaluates the approach on a problem over 50 stages, in combination with a simple implementation of $\mathcal{A}$ based on a branching process that modifies its branching probabilities to control the total expected number of scenarios.

As a last remark, we note that the optimal values $v(\mathcal{T}^m)$ of the programs $\mathcal{P}(\mathcal{T}^m)$ are a poor indicator of the relative true quality of the corresponding first-stage decisions π_1^m. Our solution approach does not use $v(\mathcal{T}^m)$ and in fact would still be valid with trees constructed by tweaking the true distributions. For instance, one may want to artificially inflate the probabilities of extreme events when constructing the scenario trees and then evaluate the resulting policies on the true distributions.

3. Mathematical Formalization

This section formalizes the supervised learning approach proposed in this paper. After summarizing the notations in §3.1, §3.2 states the generic form of the multistage stochastic program under consideration. Section 3.3 gives the generic form of a scenario-tree approximation. Section 3.4 describes the data sets extracted from an optimal solution to the scenario-tree approximation. Section 3.5 describes a Gaussian process regression method that infers from the data sets a statistical model of the mapping from information states to recourse decisions. Section 3.6 describes the optimization-based method that exploits the statistical model to output a feasible decision at each stage, given the current information state. Section 3.7 describes the simulation-based procedure for selecting a best statistical model in combination with the feasibility restoration procedure.

3.1. Notation

We follow conventions proposed in Shapiro et al. (2009, Equation (3.3)), where the random process starts with a random variable ξ_1 that has a single trivial value.

t: stage index, running from 1 to T.

$a^\top b$: inner product between two column vectors a, b of same dimension.

ξ_t: random vector observed just before taking the decision at stage t; ξ_1 is deterministic.

$\xi_{[t]} = (\xi_1, \dots, \xi_t)$: the random vectors up to stage t, that is, the history of the random process available at stage t. Shorthand for $\xi_{[t_1:t_2]} = [\xi_{t_1}^\top, \dots, \xi_{t_2}^\top]^\top$ with $t_1 = 1$ and $t_2 = t$.

x_1, $x_2(\cdot)$, …, $x_T(\cdot)$: decision policy from stage 1 to T, where the value of x_t in $\mathbb{R}^{n_t}$ is uniquely determined by the realization of $\xi_{[t]}$. For definiteness, the domain of x_t must include the support of $\xi_{[t]}$.

$f_1(\cdot), \dots, f_T(\cdot)$: convex cost functions, where $f_t(\cdot)$ depends on x_t and (if $t \ge 2$) ξ_t.

$\mathcal{X}_1, \mathcal{X}_2(\cdot), \dots, \mathcal{X}_T(\cdot)$: set-valued mappings to convex feasibility sets, in the sense that $x_t(\xi_{[t]})$ must be valued in $\mathcal{X}_t(x_{t-1}(\xi_{[t-1]}), \xi_t)$. A mapping $\mathcal{X}_t(\cdot)$ can also be expressed using another convenient representation of the information state at time t.

3.2. True Problem

The multistage stochastic program under consideration is called the true problem. The true problem is written in abstract form as

$$\begin{aligned} \min_{x_1, x_2(\cdot), \dots, x_T(\cdot)} \quad & \mathbb{E}[f_1(x_1) + f_2(x_2(\xi_{[2]}), \xi_2) + \cdots \\ & \quad + f_T(x_T(\xi_{[T]}), \xi_T)] \\ \text{subject to} \quad & x_1 \in \mathcal{X}_1,\ x_t(\xi_{[t]}) \in \mathcal{X}_t(x_{t-1}(\xi_{[t-1]}), \xi_t), \\ & \qquad t = 2, \dots, T. \end{aligned} \tag{1}$$

We also keep in mind Theorem 2.4 in Heitsch and Römisch (2011), which establishes assumptions for

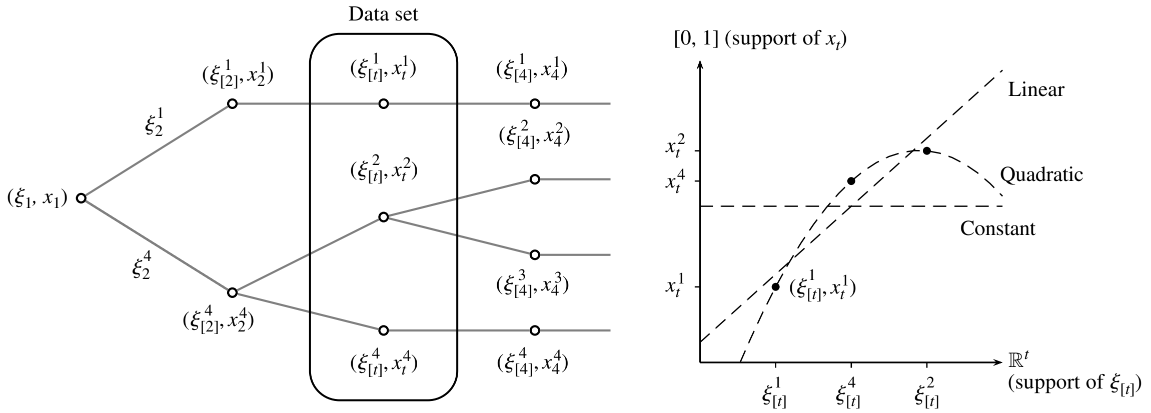


Figure 1 **Possible Models (Dashed Lines) for the Recourse Function $x_t(\cdot)$ at $t = 3$**

Notes. A data set of history-decision pairs is extracted from a scenario tree where the nodal decisions have been optimized. Illustration with parametric regression: three models of different complexity are shown, built from the same pairs $(\xi_{[t]}^k, x_t^k)$ for deriving candidate PFAs at $t = 3$.

ensuring that perturbations of the ϵ-optimal solution sets of a program such as (1) remains bounded under small perturbations of the distributions for $\xi_{[t]}$. A simple particular case of the theorem works with the following assumptions:

1. The multistage stochastic program is linear, with

$$f_1(x_1) = c_1^\top x_1,$$
$$f_t(x_t(\xi_{[t]}), \xi_t) = c_t(\xi_t)^\top x_t(\xi_{[t]}),$$
$$\mathscr{X}_1 = \{x_1 \in X_1 \colon A_{1,0} x_1 = \eta_1\},$$
$$\mathscr{X}_t(x_{t-1}(\xi_{[t-1]}), \xi_t) = \{x_t \in X_t \colon A_{t,0} x_t + A_{t,1} x_{t-1} = \eta_t(\xi_t)\},$$

for some fixed vectors c_1, η_1, some vector-valued c_t, η_t for $t \geq 2$ depending affinely on ξ_t, some nonempty, bounded, fixed sets X_t, and some fixed matrices $A_{t,0}$ for $t \geq 1$ and $A_{t,1}$ for $t \geq 2$.

2. ξ_t and $x_t(\xi_{[t]})$ have finite second moments.

3. Heitsch and Römisch (2011) also specify a perturbation-robust version of the relatively complete recourse assumption. Under small bounded perturbations of the conditional distributions for ξ_t given $\xi_{[t-1]}$, it must still be the case that the relatively complete recourse assumption holds.

4. Heitsch and Römisch (2011) also give a perturbation-robust version of assumptions ensuring that the optimal value of perturbed problems is finite and attained.

3.3. Approximation of the True Problem

The scenario-tree approximation for (1) is written in abstract form as

$$\begin{aligned} \min_{x_1^k, x_2^k, \ldots, x_T^k} \quad & \sum_{k=1}^K p^k [f_1(x_1^k) + f_2(x_2^k, \xi_2^k) + \cdots + f_T(x_T^k, \xi_T^k)] \\ \text{subject to} \quad & x_1^k \in \mathscr{X}_1,\ x_t^k \in \mathscr{X}_t(x_{t-1}^k, \xi_t^k), \\ & \qquad t = 2, \ldots, T,\ k = 1, \ldots, K, \\ & x_t^k = x_t^\ell \quad \text{for all } k, \ell \\ & \qquad\qquad \text{such that } \xi_{[t]}^k = \xi_{[t]}^\ell,\ t = 1, \ldots, T. \end{aligned} \tag{2}$$

Here $x_1^k, \ldots, x_T^k$ denote decision vectors relative to scenario k for $k = 1, \ldots, K$, $\xi_{[t]}^k$ is the history of random vectors up to stage t for scenario k, p^k is the probability of scenario k with $\sum_{k=1}^K p^k = 1$, and the additional constraints are the nonanticipativity constraints.

The discrepancy between the optimal value, solution set of (2) and the optimal value, solution set of (1) depends on the number K of scenarios, the branching structure of the tree, and the values p^k, $\xi_{[T]}^k$. Conditions for the epiconvergence of (2) to (1) (convergence of the optimal value and the optimal set for the first-stage decision) are studied in Pennanen (2005), building on the work of many others.

3.4. Extraction of State-Decision Pairs

Let us define data sets of state-decision pairs extracted from an optimal solution to (2):

$$D_t = \{(\xi_{[t]}^k, x_t^{k*})\}_{k=1}^K, \quad t = 1, \ldots, T. \tag{3}$$

Here $x_1^{k*}, \ldots, x_T^{k*}$ denote the optimized decisions relative to scenario k, given the tree. The data set D_1 is trivial because it always contains the same pair (ξ_1, x_1^*) representing the optimal first-stage decision given the tree. The data set D_T contains distinct pairs. The data sets D_t for $t = 2, \ldots, T-1$ have some duplicate entries resulting from the branching structure of the tree, assuming all necessary nonanticipativity constraints have been formulated and (2) has been solved to optimality.

We let $|D_t| \leq K$ denote the number of distinct pairs in D_t.

Clearly, the pairs in the data sets are *not* independent, identically distributed (i.i.d.) samples from a fixed but unknown distribution because the decisions x_t^k are optimized jointly given the tree. Also, the decisions x_t^{k*} are not necessarily optimal for the true problem.

Depending on the machine learning strategy, it can be beneficial to represent the information state and decision by minimal representations or, quite the opposite, to expand the representation so as to express additional features.

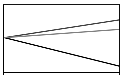
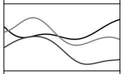
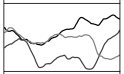
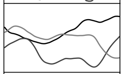
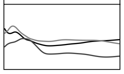
3.5. Inference of the Statistical Model for Making Decisions

Several statistical models are possible, but in the present context we find it particularly attractive to consider nonparametric models. We work with Gaussian processes (O'Hagan 1978, Rasmussen and Williams 2006). Gaussian process regression can be viewed as a Bayesian approach where one starts with a prior distribution over functions, which is then updated to a posterior distribution using observations of the value of some unknown but fixed function evaluated at different points.

A random function $\pi_t(\cdot)$ with values $y_t = \pi_t(\xi_{[t]})$, where we assume momentarily $y_t \in \mathbb{R}$, follows a Gaussian process $GP(m(\xi_{[t]}), C^\theta(\xi_{[t]}, \xi'_{[t]}))$, where $m(\cdot)$ is the mean function and $C^\theta(\cdot,\cdot)$ is the covariance function (with hyperparameters θ), if for any finite n and any fixed $\xi^1_{[t]}, \ldots, \xi^n_{[t]}$, the vector $[y^1_t, \ldots, y^n_t]^\top$ with $y^i_t = \pi_t(\xi^i_{[t]})$ follows a multivariate normal $\mathcal{N}(\mu, \Sigma)$ with $\mu_i = m(\xi^i_{[t]})$ and $\Sigma_{ij} = C^\theta(\xi^i_{[t]}, \xi^j_{[t]})$. A function $C^\theta(\cdot,\cdot)$ is an admissible covariance function if the matrix Σ is always positive semidefinite. The prior distribution on $\pi_t(\cdot)$ is determined mainly by the choice of $C^\theta(\cdot,\cdot)$. Common covariance functions have been listed in Table 1, along with examples of realizations of functions following a Gaussian process with the corresponding covariance function for some fixed hyperparameter. We use a scalar $\xi_{[t]}$ for being able to make two-dimensional plots.

As the number of observations grows, the functions drawn from the posterior would progressively adopt a common shape closer to the unknown function. Apart from the linear covariance function, all the listed basic covariance functions would permit in the limit a concentration of the posterior on an arbitrary continuous function on a convex compact set.

Table 1 Some Commonly Used Covariance Functions

Name	Expression[a] of $C^\theta(\xi_{[t]}, \xi'_{[t]})$	Samples $\pi_t(\cdot) \sim GP$
Linear	$z^\top \mathrm{Diag}(\theta) z'$ where $\theta \succeq 0$	
Gaussian	$\exp\left(-\dfrac{r^2}{2\theta^2}\right)$ where $\theta > 0$	
Matérn 3/2	$\left(1 + \dfrac{\sqrt{3}r}{\theta}\right)\exp\left(-\dfrac{\sqrt{3}r}{\theta}\right)$ $(\theta > 0)$	
Matérn 5/2	$\left(1 + \dfrac{\sqrt{5}r}{\theta} + \dfrac{5r^2}{3\theta^2}\right)\exp\left(-\dfrac{\sqrt{5}r}{\theta}\right)$	
Neural network	$\dfrac{2}{\pi}\sin^{-1}\left(\dfrac{2z^\top \Theta z'}{\sqrt{(1+2z^\top\Theta z)(1+2z'^\top\Theta z')}}\right)$ where Θ is positive definite	
		horiz. axis: $\xi_{[t]} \in \mathbb{R}$

[a] Where $r = \|\xi_{[t]} - \xi'_{[t]}\|$, $z = [1\ \xi^\top_{[t]}]^\top$, $z' = [1\ \xi'^\top_{[t]}]^\top$.

Table 2 Some Operations that Preserve Positive Semidefiniteness

Sum	$C_t(\xi_{[t]}, \xi'_{[t]}) = C^A(\xi_{[t]}, \xi'_{[t]}) + C^B(\xi_{[t]}, \xi'_{[t]})$
Product	$C_t(\xi_{[t]}, \xi'_{[t]}) = C^A(\xi_{[t]}, \xi'_{[t]}) \cdot C^B(\xi_{[t]}, \xi'_{[t]})$
Direct sum	$C_t(\xi_{[t]}, \xi'_{[t]}) = C^A(\xi_{[1:\tau]}, \xi'_{[1:\tau]}) + C^B(\xi_{[\tau+1:t]}, \xi'_{[\tau+1:t]})$
Tensor product	$C_t(\xi_{[t]}, \xi'_{[t]}) = C^A(\xi_{[1:\tau]}, \xi'_{[1:\tau]}) \cdot C^B(\xi_{[\tau+1:t]}, \xi'_{[\tau+1:t]})$
Warping	$C_t(\xi_{[t]}, \xi'_{[t]}) = C^A(g(\xi_{[t]}), g(\xi'_{[t]}))$ where $g(\cdot)$ is arbitrary

Common licit operations for combining or transforming covariance functions have also been listed in Table 2. Sophisticated covariance functions provide more flexibility for injecting prior knowledge and for improving the generalization of finite data sets.

When $y_t = \pi_t(\xi_t)$ is vector-valued, we assume that $m(\cdot)$ is vector-valued with n_t coordinates written $m_{t,i}(\cdot)$, $i = 1, \ldots, n_t$. Given $\xi_{[t]}$, we treat each coordinate $y_{t,i}$ of $\pi_t(\xi_{[t]})$ independently.

Posterior distributions are specified through the mean and covariance matrix of the function values at query points, given noisy observations of the function values at a finite number of locations. For the purpose of writing down the update formulae given the data set (3), we define the following shorthand notations:

$$x^*_{t,i}(D_t) = [x^{1*}_{t,i} \cdots x^{|D_t|*}_{t,i}]^\top \in \mathbb{R}^{|D_t|\times 1}, \quad (4)$$

$$m_{t,i}(D_t) = [m_{t,i}(\xi^1_{[t]}) \cdots m_{t,i}(\xi^{|D_t|}_{[t]})]^\top \in \mathbb{R}^{|D_t|\times 1}, \quad (5)$$

$$C^\theta_t(D_t, \xi_{[t]}) = [C^\theta_t(\xi^1_{[t]}, \xi_{[t]}) \cdots C^\theta_t(\xi^{|D_t|}_{[t]}, \xi_{[t]})]^\top \in \mathbb{R}^{|D_t|\times 1}, \quad (6)$$

$$C^\theta_t(D_t, D_t) = \begin{bmatrix} C^\theta_t(\xi^1_{[t]}, \xi^1_{[t]}) & \cdots & C^\theta_t(\xi^1_{[t]}, \xi^{|D_t|}_{[t]}) \\ \vdots & & \vdots \\ C^\theta_t(\xi^{|D_t|}_{[t]}, \xi^1_{[t]}) & \cdots & C^\theta_t(\xi^{|D_t|}_{[t]}, \xi^{|D_t|}_{[t]}) \end{bmatrix} \in \mathbb{R}^{|D_t|\times|D_t|}. \quad (7)$$

We express the samples x^{k*}_t in the data sets (3) as noisy observations of some unknown but fixed function π_t at the locations $\xi^k_{[t]}$:

$$x^{k*}_t = \pi_t(\xi^k_{[t]}) + w^k_t, \quad (8)$$

where the unknown w^k_t is an i.i.d. zero-mean Gaussian noise with covariance $\sigma^2_w I \in \mathbb{R}^{n_t \times n_t}$, the value of σ^2_w being a knob of the statistical model for x^{k*}_t.

The value of the unknown function π_t at a new query point $\xi_{[t]}$ is then described probabilistically given (3): $\pi_{t,i}(\xi_{[t]})$ follows $\mathcal{N}(\lambda^\theta_{t,i}(\xi_{[t]}), \Lambda^\theta_{t,i}(\xi_{[t]}))$, where

$$\lambda^\theta_{t,i}(\xi_{[t]}) = m_{t,i}(\xi_{[t]}) + C^\theta_t(D_t, \xi_{[t]})^\top [C^\theta_t(D_t, D_t) + \sigma^2_w I]^{-1}$$

$$\cdot\, (x^*_{t,i}(D_t) - m_{t,i}(D_t)), \tag{9}$$

$$\Lambda^\theta_{t,i}(\xi_{[t]}) = C^\theta_t(\xi_{[t]}, \xi_{[t]}) - C^\theta_t(D_t, \xi_{[t]})^\top \cdot [C^\theta_t(D_t, D_t) + \sigma^2_w I]^{-1} C^\theta_t(D_t, \xi_{[t]}), \tag{10}$$

with I denoting the identity matrix in $\mathbb{R}^{|D_t|\times|D_t|}$.

We can give to π_t in (8) the interpretation of an unknown but fixed near-optimal policy at stage t for the true problem. If we had an infinite number of samples and branchings in the scenario tree, then the infinite number of optimal nodal decisions would tend to the optimal decisions for the true problem by epiconvergence of the scenario-tree problem to the true problem (Pennanen 2005). Then, from the regression on that infinite number of samples, the distribution over the functions π_t would concentrate on a function that would represent a near-optimal recourse function at stage t, provided that the covariance function is flexible enough.

With a finite scenario tree, the target decisions x^{k*}_t are biased, and the locations $\xi^k_{[t]}$ do not fully cover the input space. We can mitigate these effects by augmenting the noise variance and by choosing different hyperparameters for the covariance function.

If we want to replicate closely the decisions optimal for (2), a small noise variance σ^2_w should be chosen that will simply act as a numerical regularizer in the inversion of the matrix $C^\theta_t(D_t, D_t)$. If we have some confidence in the prior and some mistrust in the approximation of (1) by (2), a larger noise variance could be chosen. A larger noise variance reduces the weight of the updates made to the decisions $m_{t,i}(\xi_{[t]})$ determined by the prior.

For the mean functions, it is common in the machine learning literature to set $m_t \equiv 0$. Another option would be to first solve a deterministic approximation of the stochastic program (1), typically the multistage problem on a single scenario; extract the optimized decisions x_t; and set $m_t \equiv x_t$. Update formulae also exist where m_t is replaced by a few parametric functions endowed with a noninformative prior (O'Hagan 1978).

The choice of the covariance functions C^θ_t (along with their hyperparameters θ) affects the "regularity" of the decision policies that are generated from the updated distribution. Optimal policies for the true problem may not be smooth or continuous functions of $\xi_{[t]}$, but usually (with mean functions set to 0) it is not possible to obtain discontinuous functions from Gaussian processes.

Perturbation analysis for optimization suggests, however, that near-optimal solution sets can exhibit a smoother behavior than the optimal solution sets. For instance, the optimal solution to $\arg\max_{x\in[-1,1]} \xi x$ is $\operatorname{sign}(\xi)$, which is discontinuous, but if we approximate $\operatorname{sign}(\xi)$ by the smooth function $\tanh(3\xi)$, we obtain a solution of relative accuracy 10%; with $\tanh(30\xi)$ the relative accuracy is 1% (we compute relative accuracies by evaluating numerically $R(r) = \max_{\xi\geq 0} \xi(1 - \tanh(r\xi))$, which gives the worst-case regret for a fixed r).

In the present paper, the choice of the covariance functions and hyperparameters is incorporated to a general model selection procedure based on the simulation of the decision policy on the true problem (see §3.7).

3.6. Inference of a Feasible Decision Policy

In a Bayesian framework, the estimate of the decision $x_t(\xi_{[t]}) \in \mathbb{R}^{n_t}$ is not described by a single vector, but by a predictive distribution.

We define in this section simple selection procedures that will output a single feasible decision $\tilde{x}_t(\xi_{[t]})$.

Under the Gaussian process model, the density of the predictive distribution is the density of a multivariate Gaussian with mean $\lambda^\theta_t(\xi_{[t]}) = [\lambda^\theta_{t,1}(\xi_{[t]}), \ldots, \lambda^\theta_{t,n_t}(\xi_{[t]})]^\top$ and covariance matrix $\Lambda^\theta_t(\xi_{[t]}) = \operatorname{diag}(\Lambda^\theta_{t,1}(\xi_{[t]}), \ldots, \Lambda^\theta_{t,n_t}(\xi_{[t]}))$, using (9), (10).

We can select a single feasible decision $\tilde{x}_t(\xi_{[t]})$ to be implemented by maximizing the log-likelihood of the density, subject to feasibility constraints:

$$\tilde{x}_t(\xi_{[t]}) = \arg\min_{x_t} \; (x_t - \lambda^\theta_t(\xi_{[t]}))^\top [\Lambda^\theta_t(\xi_{[t]})]^{-1} \cdot (x_t - \lambda^\theta_t(\xi_{[t]})) \tag{11}$$
$$\text{subject to } \; x_t \in \mathscr{X}_t(\tilde{x}_{t-1}(\xi_{[t-1]}), \xi_t).$$

The program (11) is essentially the implementation of a projection operator on the feasibility set $\mathscr{X}_t(\tilde{x}_{t-1}(\xi_{[t-1]}), \xi_t)$, applied to the conditional mean $\lambda^\theta_t(\xi_{[t]})$: see Figure 2.

Solving (11) after having evaluated $\lambda^\theta_t(\xi_{[t]})$, $[\Lambda^\theta_t(\xi_{[t]})]^{-1}$ induces a feasible decision policy.

Another, faster option is first to select the mean $\lambda^\theta_t(\xi_{[t]})$ and then to correct it with some fast heuristic for restoring its feasibility. The heuristic could itself have a small number of parameters θ^{heur}. The heuristic can be interpreted as being a part of the decision maker's prior on near-optimal decision policies.

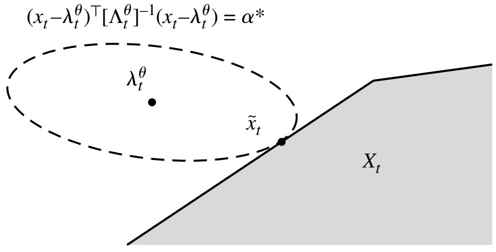

$(x_t - \lambda^\theta_t)^\top [\Lambda^\theta_t]^{-1} (x_t - \lambda^\theta_t) = \alpha^*$
λ^θ_t
$\tilde{x}_t$
X_t

Figure 2 Restoring the Feasibility of a Prediction λ^θ_t for $x_t \in \mathscr{X}_t$ by Solving (11), Where α^* Denotes the Corresponding Optimal Value

3.7. Model Selection

Ultimately, a decision policy should be selected not for its ability to explain the decisions of the scenario-tree approximation, but for its ability to output decisions leading to the best possible expected performance on the true problem.

The performance of any feasible policy $\tilde{\pi} = (\tilde{\pi}_1, \ldots, \tilde{\pi}_T)$ is actually the value of $\tilde{\pi}$ for the true multistage problem, written $v(\tilde{\pi})$. It can be estimated by Monte Carlo simulation over a large test sample of i.i.d. scenarios $\xi^{\#j} = (\xi_1^{\#j}, \ldots, \xi_T^{\#j})$, $1 \leq j \leq N$, independent of the scenarios of the scenario tree. The estimator of $v(\tilde{\pi})$ on the test sample $TS = \{\xi^{\#j}\}_{j=1}^N$ is a sample average approximation (SAA) estimator:

$$v^{TS}(\tilde{\pi}) = \frac{1}{N}\sum_{j=1}^{N}\big[f_1(\tilde{\pi}_1) + f_2(\tilde{\pi}_2(\xi^{\#j}_{[2]}), \xi^{\#j}_2) + \cdots + f_T(\tilde{\pi}_T(\xi^{\#j}_{[T]}), \xi^{\#j}_T)\big]. \qquad (12)$$

If $v(\tilde{\pi})$ is finite, the estimate (12) is an unbiased estimator of the value of $\tilde{\pi}$ on the true problem, by the strong law of large numbers. Moreover, if the objective of the true problem under the policy $\tilde{\pi}$ has a finite second moment, by the central limit theorem (12) is approximately normally distributed, with a variance that can be estimated by

$$\hat{\sigma}^2(v^{TS}(\tilde{\pi})) = \frac{1}{N}\left(\frac{1}{N-1}\sum_{j=1}^{N}[f_1(\tilde{\pi}_1) + \cdots + f_T(\tilde{\pi}_T(\xi^{\#j}_{[T]}), \xi^{\#j}_T) - v^{TS}(\tilde{\pi})]^2\right).$$

Hence, one can guarantee with an approximate level of confidence $(1-\alpha)$ that

$$v(\tilde{\pi}) \leq v^{TS}(\tilde{\pi}) + z_{\alpha/2}\hat{\sigma}(v^{TS}(\tilde{\pi})) \qquad (13)$$

with $z_{\alpha/2} = \Phi^{-1}(1-\alpha/2)$ and Φ^{-1} denoting the inverse cumulative distribution function of the standard normal distribution (Shapiro et al. 2009).

The right-hand side of (13) is a statistical performance guarantee on the true problem. Ranking different policies derived from different priors is possible on the basis of (13), although more efficient ranking and selection techniques could be employed to eliminate bad policies more rapidly (we are thankful to Alexander Shapiro for this suggestion).

4. Numerical Test

We investigate the proposed methodology numerically according to three main factors of variation in its implementation: (i) the size of the scenario tree used for approximating the true problem, relative to the size that should be used to solve the multistage problem accurately; (ii) the choice of the covariance function of the Gaussian processes that determines how the decisions extracted from the scenario tree are generalized to new information states; and (iii) the choice of the feasibility restoration procedure, which plays a role if the predicted decisions are not feasible.

The decision policies derived from these experiments are evaluated according to two criteria: (i) the quality of the decision policy, relative to the best performance attainable for the true problem and (ii) the computational complexity of simulating the policy.

The experiments are implemented in Matlab and the programs are solved with cvx (Grant and Boyd 2008, 2009).

4.1. Test Problem

In the spirit of a stylized application presented in Shapiro et al. (2009, §1.3.3), we consider a four-stage assembly product problem:

$$\min_{\substack{q_1,\\ q_2(\cdot),\, Y_2(\cdot),\\ q_3(\cdot),\, Y_3(\cdot),\\ q_4(\cdot)}} \quad \mathbb{E}\left[c_1^\top q_1 + \sum_{t=2}^{4} c_t^\top q_t(\xi_{[t]})\right]$$

$$\begin{aligned}
\text{subject to}\quad & q_1 \in \mathscr{X}_1 = \{q_1 \in \mathbb{R}^{12}\colon q_{1,i} \geq 0\},\\
& (q_2(\xi_{[2]}), Y_2(\xi_{[2]})) \in \mathscr{X}_2(q_1)\\
&\quad = \Big\{(q_2, Y_2) \in \mathbb{R}^8 \times \mathbb{R}^{12\times 8}\colon\\
&\qquad A_{2,ij} q_{2,j} \leq Y_{2,ij},\ \sum_j Y_{2,ij} \leq q_{1,i},\\
&\qquad\qquad q_{2,i},\ Y_{2,ij} \geq 0\Big\},\\
& (q_3(\xi_{[3]}), Y_3(\xi_{[3]})) \in \mathscr{X}_3(q_2(\xi_{[2]}))\\
&\quad = \Big\{(q_3, Y_3) \in \mathbb{R}^5 \times \mathbb{R}^{8\times 5}\colon\\
&\qquad A_{3,ij} q_{3,j} \leq Y_{3,ij},\ \sum_j Y_{3,ij} \leq q_{2,i}(\xi_{[2]}),\\
&\qquad\qquad q_{3,i},\ Y_{3,ij} \geq 0\Big\},\\
& q_4(\xi_{[4]}) \in \mathscr{X}_4(q_3(\xi_{[3]}), \xi_{[4]})\\
&\quad = \big\{q_4 \in \mathbb{R}^5\colon q_{4,i} \leq \max\{0, b_i^\top \xi_{[4]}\}\\
&\qquad \overset{\text{def}}{=} \eta_i(\xi_{[4]}),\ 0 \leq q_{4,i} \leq q_{3,i}(\xi_{[3]})\big\},
\end{aligned} \qquad (14)$$

with $\xi_1 \equiv 1$, and ξ_2, ξ_3, ξ_4 i.i.d. random variables each following the standard normal distribution. The problem data are given in the appendix.

In this resource allocation problem, ξ_2 and ξ_3 represent observable factors that contribute to demands $\eta_i(\xi_{[4]})$ revealed at the last stage, for various end products i. The decisions $q_{t,j}$ represent quantities of a component j to be produced at stage t, by assembling components produced at the previous stage. The decisions $Y_{t,ij}$, arranged in a matrix, represent the quantity of component i (produced at stage $t-1$) allocated to the production of component j (produced at stage t), where the minimal quantity of i per unit of j is specified by $A_{t,ij} \geq 0$. The cost coefficients c_t have nonnegative components at stages 1–3 but are negative at stage 4, for representing revenue drawn from selling quantities of end products at most equal to the demand or inventory levels.

The size of the test problem has been fit to the numerical experiments to be conducted. For benchmarking, we simulate pure multistage stochastic programming decision processes that work by instantiating and solving a new version of (14) at each decision stage, over the remaining horizon, with the previous decisions fixed at their implemented value. Obtaining the benchmark values takes many hours of computation on a single processor (§4.4), but the simulations could run easily in parallel once the common first-stage decision has been computed.

The numerical parameters of the test problem (see appendix) have been chosen by selecting, among randomly generated sets of parameters, a set of parameters that "maximizes" the value of the multistage model. The value of the multistage model (Huang and Ahmed 2009) is the difference (in absolute value) between the optimal value of the multistage model (14) and the optimal value of the corresponding two-stage model, where q_2, Y_2, and q_3, Y_3 in (14) are incorporated to the first stage and $q_4(\xi_{[4]})$ is chosen at the second stage with $\xi_{[4]}$ revealed at once. The two-stage model does not exploit the opportunity of adapting the production plan to intermediate observations available before the demand is fully revealed.

4.2. Studied Variants in the Learned Policies

The policy function approximation to be learned is made of two components, the statistical model (Gaussian processes) and the feasibility restoration procedure. This section describes the variants that we have considered for the tests.

4.2.1. Covariance Functions. We report results obtained with covariance functions of the form

$$C_t^{\theta}(\xi_{[t]}^k, \xi_{[t]}^{\ell}) = \exp\left\{ -\left[\sum_{\tau=1}^{t} (g(\xi_\tau^k) - g(\xi_\tau^{\ell}))^2 \right] \Big/ (2\theta^2) \right\} \qquad (15)$$

for two choices of the function $g(\cdot)\colon \mathbb{R} \to \mathbb{R}$ (the warping transform in Table 2). In the first variant, $g(\cdot)$ is reduced to the identity function. Hence (15) is a Gaussian covariance function with bandwidth $\theta > 0$. In the second variant, $g(\cdot) = \Phi(\cdot)$, the cumulative distribution function of the standard normal distribution. Doing so allows us to emulate the effect of a nonconstant bandwidth.

4.2.2. Feasibility Restoration. Feasible decisions are generated by completing the Gaussian process model by the generic projection method (11). The program (11) has no parameter to tune.

We also study the behavior of a feasibility restoration heuristic well adapted to the test problem. The heuristic depends on some *priority order* over the coordinates j of the vectors $q_{t,j}$ in (14). It consists in creating inventory variables s_i and initializing them to the values $q_{t-1,i}$ for all i, then trying to reach the quantities $\lambda_{t,j}^{\theta}$ of the Gaussian model by consuming the products i in the needed proportions. Namely,

For all i, set $s_i = q_{t-1,i}$.
For all j considered sequentially according to a
 prespecified order σ_t, (16)
 define $\bar{q}_{t,j} = \min_i \{ s_i / A_{t,ij}\colon A_{t,ij} > 0 \}$;
 set $q_{t,j} = \min\{\lambda_{t,j}^{\theta}, \bar{q}_{t,j}\}$;
 and for all i, replace s_i by $s_i - A_{t,ij} q_{t,j}$.

The priority orders σ_t are viewed as the parameters of the heuristic. We generate the priority orders randomly by sampling in the space of permutations.

4.3. Scenario-Tree Approximations

In this section we describe how the scenario trees are built and how the shrinking-horizon procedure for out-of-sample validation is implemented. The shrinking-horizon procedure is the benchmark against which the learned policies will be compared.

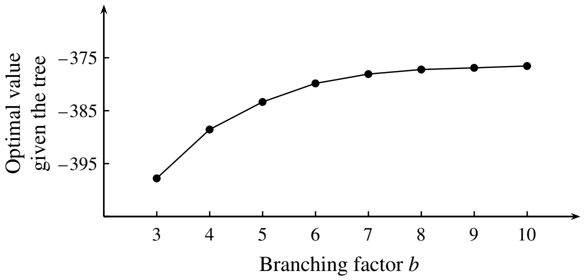
4.3.1. Method for Constructing the Scenario Trees. We considered scenario trees with a uniform branching factor b. We used an optimal quantization approach for choosing the b discrete values for ξ_t and assigning to them probability masses (Pages and Printems 2003). In a nutshell, this approach works by selecting values $\xi_t^{(i)}$ that minimize the quadratic distortion

$$D^2(\{\xi_t^{(i)}\}_{i=1}^{b}) = \mathbb{E}_{\xi_t}\left\{ \min_{1 \leq i \leq b} \|\xi_t^{(i)} - \xi_t\|^2 \right\},$$

and then evaluates probabilities $p^{(i)}$ by integrating the density of the distribution of ξ_t over the cells of the Voronoi partition induced by the points $\xi_t^{(i)}$.

A scenario tree on T stages has b^{T-1} scenarios (exponential growth). By solving scenario-tree approximations on trees with increasing branching factors, we determined that the test problem could be solved to a reasonable accuracy with a branching factor $b = 10$ (Figure 3). The solving time grows exponentially with b.

4.3.2. Value of the Multistage Model. The optimal value of the multistage model is about -375 (corresponding to a net profit); see Figure 3. This value should be compared with the optimal value of the



Branching factor b	Time (in seconds)
3	5.0
4	8.0
5	17.2
6	40.7
7	79.8
8	177.5
9	353.5
10	670.6

Figure 3 (Left) Convergence of the Optimal Value of the Approximation of (14) **Based on a Scenario Tree of Branching Factor** b. **(Right) Time for Solving the Corresponding Programs**

two-stage model. Our estimate for it is −260. The value of the multistage model over the two-stage model is thus in theory an expected profit increase of about 45%.

We recall that according to Birge and Louveaux (1997), there does not seem to be a structural property in multistage models that would guarantee a large value over their deterministic or two-stage counterpart; the value is very dependent on the numerical parameters.

4.4. Computational Results for the Benchmark Policies

We generate a test sample of $N = 10{,}000$ i.i.d. scenarios $\xi^{\#j}$ that we use for estimating the value of any policy $\tilde{\pi}$. Given a policy $\tilde{\pi}$, we simulate it on each scenario $\xi^{\#j}$ and then estimate its value by (12).

Let $\pi^{\text{ref},b}$ be the policy such that: $\pi_1^{\text{ref},b}$ is set to the solution q_1 of the program already solved given b (Figure 3); $\pi_4^{\text{ref},b}$ has values given by $q_{4,i}(\xi_{[4]}^{\#j}) = \min\{q_{3,i}(\xi_{[3]}^{\#j}), \eta_i(\xi_{[4]}^{\#j})\}$ with $q_{3,i}(\xi_{[3]}^{\#j})$ from $\pi_3^{\text{ref},b}$; and $\pi_t^{\text{ref},b}(\xi_{[t]}^{\#j})$ for $t = 2, 3$ is set to the stage-t solution of a new scenario-tree approximation of (14), given $\xi_{[t]}^{\#j}$ and $\pi_1^{\text{ref},b}, \ldots, \pi_{t-1}^{\text{ref},b}(\xi_{[t-1]}^{\#j})$, where the new tree approximates $\xi_{t+1}, \ldots, \xi_T$ given $\xi_{[t]}^{\#j}$ by the method described in §4.3.1 with branching factor b but with branchings only beyond stage t (because $\xi_{[t]} = \xi_{[t]}^{\#j}$ is known).

To investigate the effect of the size of the scenario tree on the policy, we have tested policies $\pi^{\text{ref},b}$ using $b = 3$, 5, 7. The result of these simulations is presented in Figure 4 for the performance of the policies (curve "Benchmark") and in Table 3 for the computation time on a single processor (row "Benchmark").

4.5. Computational Results for Learned Policies

We have reported on Figure 4 the results of simulations on the 10,000 scenarios for three variants:

1. GP-1: covariance function (15) with $g(z) = z$, feasibility restoration procedure (11).
2. GP-2: covariance function (15) with $g(z) = \Phi(z)$, feasibility restoration procedure (11).
3. GP-3: covariance function (15) with $g(z) = \Phi(z)$, feasibility restoration procedure (16).

Each variant was tested on the three data sets collecting the history-decision pairs for $t = 2$, 3 from a scenario tree with branching factor $b = 3$, 5, 7 (problems of Figure 3). For $t = 1$ we use the optimal decision q_1 given b, and for $t = 4$ we use $q_{4,i}(\xi_{[4]}^{\#j}) = \min\{q_{3,i}(\xi_{[3]}^{\#j}), \eta_i(\xi_{[4]}^{\#j})\}$. The determination of the best hyperparameters for each variant was made by direct search, treating each value for the hyperparameters as a possible model for the decision policy.

On the test problem we have considered, it seems that the simple program (11) introduces a large computational overhead. The simulation times of the models GP-1 and GP-2 are only 1.5 to 6 times faster than the benchmark method (Table 3). One possible explanation is that the scenario-tree approximations built on the remaining horizon have few scenarios and thus are not really much more difficult to solve than the myopic program (11). When we replace (11) by the heuristic (16) in GP-3, we obtain a very important speed-up of the simulations (Table 3) for a relatively small loss of performance with respect to the benchmark (Figure 4) and the theoretical best value of the multistage program (Figure 3 with $b = 10$).

For the GP policies, the simulation times do not vary much with b. In GP-1 and GP-2, most of the time is spent on (11), which is independent of the size of

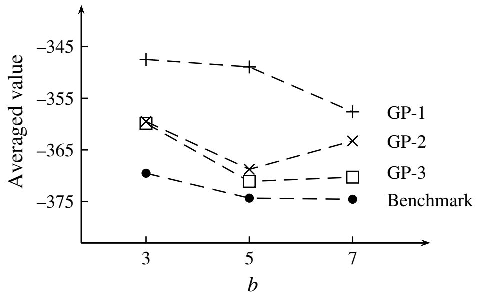


Figure 4 Simulation of Policies on 10,000 Scenarios: Averaged Value

Table 3 Simulation of Policies on 10,000 Scenarios: Computation Time

	CPU time (in seconds)		
Policy	$b=3$	$b=5$	$b=7$
Benchmark	17,000	31,000	65,000
GP-1	11,000	11,000	12,000
GP-2	12,000	12,000	12,000
GP-3	10	10	10

the data sets. In GP-3, only (9) is used; the matrix inversions are done once, and Matlab seems to treat the remaining matrix-vector multiplications in (9) with a very similar speed.

From these experiments, we conclude that the most attractive forms of learned policies for doing simulations on a large number of scenarios are those that eliminate completely the calls to a solver, even for solving very simple programs. Doing so can be achieved by setting up a feasibility restoration heuristic that need not be very clever because the implemented decisions depend mostly on the predictions of the Gaussian process model.

5. Application: Optimal Selection of Scenario Trees

This section presents our solution approach to multistage problems over long horizons, based on the selection of random scenario trees by out-of-sample validation. An overview of the selection algorithm and of our general methodology is given in Figure 5. Section 5.1 makes the stylized example of §2 more concrete. Section 5.2 presents an algorithm for generating random branching structures that turned out to be well adapted to the problem of §5.1. Section 5.3 presents the computational study. This material is also presented in Defourny et al. (2012). Section 5.4 considers the number of random trees to sample.

Given a multistage stochastic program over T stages:

1. Generate M random trees of approximately N scenarios.
2. Solve the M stochastic programs corresponding to the trees.
3. Create the M data sets of history-decision pairs taken from the trees.
4. Learn N^π policies per data set (by the methods of §3).
5. Simulate the $M \cdot N^\pi$ policies on a test sample of N' i.i.d. scenarios
6. Select the best policy and the corresponding "best" tree.

[7.] Ideally: Improve the hyperparameters of the tree generation algorithm.

Figure 5 Optimal Selection of Scenario Trees: Algorithm and Methodology

5.1. Studied Problem

Consider the following problem, studied by Hilli and Pennanen (2008) and Küchler and Vigerske (2010), where ρ is a risk-aversion parameter and η a budget parameter:

$$\min_{x_1(\cdot),\ldots,x_T(\cdot)} \rho^{-1}\log\mathbb{E}\left\{\exp\left\{-\rho\sum_{t=1}^{T}\xi_t x_t(\xi_{[t]})\right\}\right\}$$

$$\text{subject to } x_t(\xi_{[t]}) \in \mathscr{X}_t(x_1(\xi_{[1]}),\ldots,x_{t-1}(\xi_{[t-1]})), \quad t=1,\ldots,T, \tag{17}$$

$$\mathscr{X}_t(x_1(\xi_{[1]}),\ldots,x_{t-1}(\xi_{[t-1]})) = \left\{x_t\in\mathbb{R}\colon \sum_{\tau=1}^{t-1}x_\tau(\xi_{[\tau]})+x_t\le\eta,\ 0\le x_t\le 1\right\}.$$

Here $\xi_1,\ldots,\xi_T$ corresponds to a price process centered on a strike price κ so that a positive revenue is obtained at stage t if $\xi_t > 0$ and $x_t(\xi_{[t]}) > 0$. The process is driven by a geometric Brownian motion and can be described by the following equations:

$$\xi_t = s_t - \kappa, \quad s_t = s_{t-1}\exp\{\sigma\epsilon_t - \sigma^2/2\} \quad \text{with } s_0 = \kappa, \tag{18}$$

where $\sigma^2 = 0.07$, $\kappa = 1$, and $\epsilon_1,\ldots,\epsilon_t$ are i.i.d. from the standard normal distribution. (Because ξ_1 is truly stochastic, the problem is over T recourse stages. We could introduce a trivial constant first-stage decision $x_0 = 0$ associated to $\xi_0 \equiv s_0 - \kappa = 0$, so strictly speaking the multistage model is over $T+1$ stages with $t=0$ corresponding to the first stage.)

When ρ tends to 0, the program (17) becomes linear, and for this case an optimal policy is the simple bang-bang policy: $x_t = 1$ if $\xi_t > 0$, and $t > T-\eta$; $x_t = 0$ otherwise.

Our goal is to solve (17) on $T = 52$ for various values of ρ and η. On long horizons, it is out of the question to consider scenario trees with uniform branching factors (with $b = 2$ we have already $2^{52} = 4.5\cdot 10^{15}$ scenarios).

Interestingly, the decisions optimal for scenario-tree approximations of (17) turn out to be very dependent on the branching structure of the tree. When a branching is missing in one scenario of the tree, a deterministic vision of the future is induced for that scenario from the stage of the missing branching to the stage where a branching is finally encountered. This will not hurt if the value of the multistage model on this part of the scenario and onward is negligible, but we cannot know that in advance (that is, prior to having computed an optimal policy that solves (17) or at least having determined its structure).

There does not seem to be much advantage in devoting computational resources to an optimization of the branching structure of the tree because at the end of the day we would still be unable to estimate how realistic the optimal value of the approximation is with respect to the true optimal value or with respect to a binary tree of 2^{52} scenarios.

Motivated by these considerations, we propose to generate branching structures purely randomly. The approach makes sense only if we can estimate the value of the approximation for the true problem without an optimistic bias. This is in turn possible by simulating a decision policy learned from the solution to the scenario-tree approximation. With a learning and simulation procedure that is fast enough, we can score several scenario trees and thus in essence implement a Monte Carlo search algorithm in the high-dimensional space of branching structures.

5.2. Random Generation of Branching Structures

Based on various numerical experiments for solving (17), the algorithm described in Figure 6 has been found to work well for generating, with a sufficient probability, branching structures leading ultimately to good decision policies.

The algorithm produces branching structures leading to trees having approximately N scenarios in the following sense. Assume that the number ν_{T-1} of existing nodes at depth $T-1$ is large. From each node, create one or two successor nodes randomly (refer to step 3 in Figure 6). By the independence of the random variables Z_j that determine the creation of one or two successor nodes, and by the weak law of large numbers, the created random number of nodes at depth T is approximately equal to

$$\begin{aligned}\nu_T &= \nu_{T-1}(2\cdot r_{t-1}+1\cdot(1-r_{t-1})) = \nu_{T-1}(1+r_{t-1})\\ &= \nu_{T-1}(1+(1/\nu_{T-1})(N-1)/T) = \nu_{T-1}+(N-1)/T.\end{aligned}$$

Iterating this recursion yields $\nu_T = \nu_0 + T(N-1)/T = N$. To establish the result, we have neglected that when ν_{t-1} is small, the random value of ν_t conditionally to ν_{t-1} should not be approximated by the conditional mean of ν_t, as done in the recursive formula. The error affects mostly the first depth levels of the tree under development. We have found, by generating random trees and estimating the expectation and variance of the number of leafs, that the error had a small effect in practice.

```
Given $N$ (desired approximate total number of scenarios):
  1. Create a root node (depth 0). Set $t = 0$.
  2. Set $\nu_t$ to the number of nodes at depth $t$.
       Set $r_t = (1/\nu_t)(N-1)/T$.
  3. For each node $j$ of depth $t$:
     Draw $Z_j$ uniformly in the interval $[0, 1]$.
     If $Z_j \le r_t$, append 2 children nodes to node $j$
       (binary branching).
     If $Z_j > r_t$, append 1 child node to node $j$
       (no branching).
  4. If $t < T-1$, increment $t$ and go to step 2.
     Otherwise, return the branching structure.
```

Figure 6 **Random Generation of Scenario-Tree Branching Structures**

5.3. Computational Results

We have considered three sets of 25 scenario trees generated randomly over the horizon $T = 52$: the first set with $N = T$, the second set with $N = 5T$, and the third set with $N = 25T$. The random structures are generated by the algorithm described in Figure 6. The scenarios use values of ξ_t generated randomly according to (18).

The inference of policies from the history-decision pairs of a tree starts by transforming the history $\xi_{[t]}$ to a more compact representation for the learning algorithm. We use a change of functions $x_t(\cdot) = \min\{1, \eta - \sum_{\tau=1}^{t-1} x_\tau(\cdots)\} y_t(\cdot)$ and we learn the functions $y_t(\cdot)$ instead of $x_t(\cdot)$. Feasibility is ensured whenever the learned models $\hat{y}_t(\cdot)$ have values in $[0, 1]$. These conditions are enforced at the level of the regression, so that the feasibility restoration step is bypassed (in fact, the change of functions plays its role). For details, see Defourny (2010).

The computational results are summarized in Table 4 for the accuracy (the best values are indicated in bold) and in Table 5 for the overall computational complexity of the approach that involves generating the 25 random trees, solving them, and simulating 5 candidate policies per tree on 10,000 new scenarios. The reported times are relative to a Matlab implementation, run on a single processor, but the nature of our randomized approach makes it very easy to parallelize. As a simple benchmark, we use the bang-bang policy, which is optimal for the risk-neutral case $\rho = 0$ (but far from optimal when $\rho > 0$).

It is somewhat surprising to see that multiplying the number of scenarios by 25 does not translate to significantly better results, as shown by comparing the column $N = 52$ to the column $N = 1{,}300$ in Table 4. Note, however, that the results with $N = 52$ are obtained *for a particular tree* of the set of 25. Most of the time, the results on trees with $N = 52$ are poor. Also, having 52 scenarios or 1,300 in the tree is equally

Table 4 **Value of the Best Policies Found for Instances of** (17) **with** $T = 52$

Problem		Simulation on 10,000 new scenarios: Average value				
				Best learned policies, for 3 tree sizes		
ρ	η	Benchmark		$N = 52$	$N = 260$	$N = 1{,}300$
0	2	−**0.40**	Optimal	−0.34	−0.32	−0.39
	6	−**1.19**		−1.07	−1.03	−1.18
	20	−**3.64**		−3.59	−3.50	−3.50
0.25	2	−**0.34**	Suboptimal	−0.32	−0.31	−0.33
	6	−0.75		−0.78	−0.78	−**0.80**
	20	−1.46		−1.89	−**1.93**	−1.91
1	2	−0.22		−**0.25**	−0.22	−0.24
	6	−0.37		−0.53	−0.53	−**0.54**
	20	−0.57		−0.96	−**0.98**	−0.96

Table 5 Computation Times

Problem		Total CPU time (in seconds)		
ρ	η	$N = 52$	$N = 260$	$N = 1{,}300$
0	2	415	551	1,282
	6	435	590	1,690
	20	465	666	1,783
0.25	2	460	780	2,955
	6	504	1,002	4,702
	20	524	1,084	5,144
1	2	485	986	4,425
	6	524	1,095	5,312
	20	543	1,234	6,613

terribly small compared to the exponential number required to solve the program on $T = 52$ accurately.

5.4. On the Required Number of Random Scenario Trees

Finally, we note that if there exists a randomized algorithm able to generate with probability $p^g > 0$ a tree from which a "good" decision policy can be learned (we discuss the sense of "good" below), then the number M of trees that have to be generated independently for ensuring with probability δ that at least one of them is "good" is equal to

$$M(\delta) = \left\lceil \frac{\log(1-\delta)}{\log(1-p^g)} \right\rceil. \tag{19}$$

For instance, if the randomized algorithm generates a good tree with probability 0.01, we need a set of 300 random trees to obtain a good one with probability 0.95.

The sense of "good" can be made precise in several ways: by defining an aspiration level with respect to a lower bound on the true value of the multistage program, obtained for instance with the techniques of Mak et al. (1999); by defining an aspiration level with respect to a benchmark solution that the decision maker tries to improve; or by defining aspiration levels with respect to risk measures aside from the expectation.

Indeed, it is possible to compare policies on the basis of the empirical distribution of their cumulated cost on a large test sample of independent scenarios.

6. Conclusions

This paper has presented an approach for inferring decision policies (decision rules) from the solution of scenario-tree approximations to multistage stochastic programs. Precise choices for implementing the approach have been presented in a Bayesian framework, leading to a nonparametric approach based on Gaussian processes. The sensitivity of the approach has been investigated on a particular problem through computational experiments.

The inference of decision policies could be a useful tool to calibrate scenario-tree generation algorithms. This line of research has been followed by developing a solution strategy that works by generating scenario trees randomly and then ranking them using the best policy that can be inferred from their solution. Further work could be useful for identifying randomized algorithms likely to generate good scenario trees. If these algorithms exist, a solution strategy based on them could fully leverage the computing power of current supercomputer architectures.

Appendix. Numerical Parameters

The value of the numerical parameters in the test problem (14) are given here.

$$\begin{aligned} c_1 = {} & [0.25\ 1.363\ 0.8093\ 0.7284\ 0.25\ 0.535\ 0.25\ 0.25 \\ & 0.25\ 0.4484\ 0.25\ 0.25]^\top \end{aligned}$$

$$c_2 = [2.5\ 2.5\ 2.5\ 2.5\ 13.22\ 2.5\ 3.904\ 2.5]^\top$$

$$c_3 = [3.255\ 2.5\ 2.5\ 8.418\ 2.5]^\top$$

$$c_4 = -[21.87\ 98.16\ 31.99\ 10\ 10]^\top$$

$$b_1 = [13.9\ 9.708\ 2.14\ 4.12]^\top$$

$$b_2 = [12.86\ 9.901\ 6.435\ 7.446]^\top$$

$$b_3 = [18.21\ 7.889\ 3.2\ 2.679]^\top$$

$$b_4 = [10.14\ 4.387\ 9.601\ 4.399]^\top$$

$$b_5 = [17.21\ 4.983\ 7.266\ 9.334]^\top$$

$$A_2 = \begin{bmatrix} 0.4572 & 0 & 4.048 & 0 & 0 & 0 & 0.8243 & 11.37 \\ 0 & 0 & 0.7674 & 0.5473 & 0.3776 & 0 & 0 & 0 \\ 0.4794 & 0 & 0.4861 & 1.223 & 0 & 1.475 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5114 & 0.3139 & 0 & 0 \\ 0 & 12.29 & 1.378 & 0 & 0.3748 & 0.4554 & 0 & 0 \\ 0.7878 & 0 & 0.293 & 1.721 & 0 & 0 & 0 & 0 \\ 1.504 & 0.4696 & 0.248 & 0 & 0.1852 & 0 & 0.3486 & 0 \\ 0 & 1.204 & 0 & 0.7598 & 0.452 & 0 & 0 & 0 \\ 0 & 0 & 0.2515 & 0.3753 & 0.6249 & 0 & 1.248 & 0 \\ 1.545 & 0 & 0 & 0 & 0 & 0 & 0.2732 & 0 \\ 0 & 0 & 0 & 0.6597 & 0 & 2.525 & 0 & 0 \\ 0 & 0 & 1.595 & 0 & 0 & 1.51 & 1.041 & 0.9847 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 0 & 1.223 & 0.6367 & 0 & 0 \\ 0 & 0 & 0 & 1.111 & 0 \\ 0 & 0 & 0.4579 & 0 & 0 \\ 0 & 0.1693 & 0.6589 & 0 & 0 \\ 0.5085 & 2.643 & 0 & 0 & 0 \\ 0.4017 & 0 & 0 & 0 & 0 \\ 0 & 0.7852 & 85.48 & 0 & 0 \\ 0 & 0 & 0 & 0.806 & 0.5825 \end{bmatrix}.$$

Acknowledgments

This paper presents research results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office. The scientific responsibility rests with its authors. Damien Ernst is a research associate of the Belgian FNRS of which he acknowledges the financial support. This work was carried out while Boris Defourny was affiliated with the Department of Electrical Engineering and Computer Science of the University of Liège. The authors are very grateful to Warren B. Powell for comments and suggestions that helped improve the presentation of the paper.

References

Abbeel P, Ng A (2004) Apprenticeship learning via inverse reinforcement learning. *Proc. 21st Internat. Conf. Machine Learn.* (*ICML 2004*), (ACM, New York), 1–8.

Bertsekas DP (2005) *Dynamic Programming and Optimal Control*, 3rd ed. (Athena Scientific, Belmont, MA).

Berstsekas DP, Tsitsiklis J (1996) *Neuro-Dynamic Programming* (Athena Scientific, Belmont, MA).

Birge JR (1997) State-of-the-art-survey—Stochastic programming: Computation and applications. *INFORMS J. Comp.* 9(2):111–133.

Birge JR, Louveaux F (1997) *Introduction to Stochastic Programming* (Springer, New York).

Busoniu L, Babuska R, De Schutter B, Ernst D (2010) *Reinforcement Learning and Dynamic Programming Using Function Approximators* (CRC Press, Boca Raton, FL).

Chiralaksanakul A (2003) Monte Carlo methods for multi-stage stochastic programs. Ph.D. thesis, University of Texas at Austin, Austin.

Coates A, Abbeel P, Ng A (2008) Learning for control from multiple demonstrations. *Proc. 25th Internat. Conf. on Machine Learn.* (*ICML 2008*) (ACM, New York), 144–151.

Defourny B (2010) Machine learning solution methods for multistage stochastic programming. Ph.D. thesis, University of Liège, Liège, Belgium.

Defourny B, Ernst D, Wehenkel L (2009) Bounds for multistage stochastic programs using supervised learning strategies. *Proc. 5th internat. Conf. Stochastic Algorithms: Foundations and Applications* (*SAGA 2009*), Lecture Notes in Computer Science, Vol. 5792. (Springer-Verlag, Berlin), 61–73.

Defourny B, Ernst D, Wehenkel L (2012) Multistage stochastic programming: A scenario tree based approach to planning under uncertainty. Morales EF, Sucar LE, Hoey J, eds. *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions* (IGI Global, Hershey, PA), 97–143.

Dempster MAH, Pflug G, Mitra G, eds. (2008) *Quantitative Fund Management* (Financial Math. Series, Chapman & Hall/CRC, Boca Raton, FL).

Dupacova J, Consigli G, Wallace SW (2000) Scenarios for multistage stochastic programs. *Ann. Oper. Res.* 100(1–4):25–53.

Frauendorfer K (1996) Barycentric scenario trees in convex multistage stochastic programming. *Math. Programming* 75(2):277–294.

Grant M, Boyd S (2008) Graph implementations for nonsmooth convex programs. Blondel V, Boyd S, Kimura H, eds. *Recent Advances in Learning and Control—A Tribute to M. Vidyasagar*, Lecture Notes in Control and Information Systems, Vol. 371 (Springer, New York), 95–110.

Grant M, Boyd S (2009) CVX: Matlab software for disciplined convex programming (Web page and software). Accessed February 28, 2009 http://cvxr.com/cvx/.

Hastie T, Tibshirani R, Friedman J (2009) *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed. (Springer, New York).

Heitsch H, Römisch W (2009) Scenario tree modeling for multistage stochastic programs. *Math. Programming* 118(2):371–406.

Heitsch H, Römisch W (2011) Stability and scenario trees for multistage stochastic programs. Infanger G, ed. *Stochastic Programming—The State of the Art, In Honor of George B. Dantzig* (Springer, New York), 139–164.

Hilli P, Pennanen T (2008) Numerical study of discretizations of multistage stochastic programs. *Kybernetika* 44(2):185–204.

Høyland K, Wallace SW (2001) Generating scenario trees for multistage decision problems. *Management Sci.* 47(2):295–307.

Huang K, Ahmed S (2009) The value of multistage stochastic programming in capacity planning under uncertainty. *Oper. Res.* 57(4):893–904.

Kallrath J, Pardalos PM, Rebennack S, Scheidt M, eds. (2009) *Optimization in the Energy Industry* (Springer-Verlag, Berlin).

Kouwenberg R (2001) Scenario generation and stochastic programming models for asset liability management. *Eur. J. Oper. Res.* 134(2):279–292.

Küchler C, Vigerske S (2010) Numerical evaluation of approximation methods in stochastic programming. *Optimization* 59(3):401–415.

Mak W-K, Morton DP, Wood RK (1999) Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Oper. Res. Let.* 24(1–2):47–56.

Mulvey JM, Kim WC (2011) Multistage financial planning models: Integrating stochastic programs and policy simulators. Infanger G, ed. *Stochastic Programming—The State of the Art, In Honor of George B. Dantzig* (Springer, New York), 257–276.

Nesterov Y, Vial J-P (2008) Confidence level solutions for stochastic programming. *Automatica* 44(6):1559–1568.

O'Hagan A (1978) Curve fitting and optimal design for prediction. *J. Roy. Statist. Soc.* 40(1):1–42.

Pages G, Printems J (2003) Optimal quadratic quantization for numerics: The Gaussian case. *Monte Carlo Methods Appl.* 9(2):135–166.

Pennanen T (2005) Epi-convergent discretizations of multistage stochastic programs. *Math. Oper. Res.* 30(1):245–256.

Pennanen T (2009) Epi-convergent discretizations of multistage stochastic programs via integration quadratures. *Math. Programming* 116(1):461–479.

Peters J, Schaal S (2008) Natural actor-critic. *Neurocomputing* 71(7–9):1180–1190.

Powell WB (2011) *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, 2nd ed. (Wiley, Hoboken, NJ).

Rasmussen CE, Williams CKI (2006) *Gaussian Processes for Machine Learning* (MIT Press, Cambridge, MA).

Rockafellar RT, Wets RJ-B (1991) Scenarios and policy aggregation in optimization under uncertainty. *Math. Oper. Res.* 16(1):119–147.

Shapiro A (2003) Inference of statistical bounds for multistage stochastic programming problems. *Math. Methods Oper. Res.* 58(1):57–68.

Shapiro A, Dentcheva D, Ruszczyński A (2009) *Lectures on Stochastic Programming: Modeling and Theory* (MPS-SIAM Series on Optimization, SIAM, Philadelphia).

Sutton RS, Barto AG (1998) *Reinforcement Learning, an Introduction* (MIT Press, Cambridge, MA).

Syed U, Bowling M, Schapire RE (2008) Apprenticeship learning using linear programming. *Proc. 25th Internat. Conf. Machine Learn.* (*ICML 2008*) (Omni Press, Madison, WI), 1032–1039.

Szepesvári C (2010) *Algorithms for Reinforcement Learning* (Morgan & Claypool Publishers).

Thénié J, Vial J-P (2008) Step decision rules for multistage stochastic programming: A heuristic approach. *Automatica* 44(6):1569–1584.

Wallace SW, Ziemba WT, eds. (2005) *Applications of Stochastic Programming* (MPS-SIAM Series on Optimization, SIAM, Philadelphia).