Multistage Stochastic Programming: A Scenario Tree Based Approach to Planning under Uncertainty

Boris Defourny, Damien Ernst, and Louis Wehenkel

University of Liège, Systems and Modeling, B28,
B-4000 Liège, Belgium
{Boris.Defourny,dernst,L.Wehenkel}@ulg.ac.be

Abstract. In this chapter, we present the multistage stochastic programming framework for sequential decision making under uncertainty and stress its differences with Markov Decision Processes. We describe the main approximation technique used for solving problems formulated in the multistage stochastic programming framework, which is based on a discretization of the disturbance space. We explain that one issue of the approach is that the discretization scheme leads in practice to ill-posed problems, because the complexity of the numerical optimization algorithms used for computing the decisions restricts the number of samples and optimization variables that one can use for approximating expectations, and therefore makes the numerical solutions very sensitive to the parameters of the discretization. As the framework is weak in the absence of efficient tools for evaluating and eventually selecting competing approximate solutions, we show how one can extend it by using machine learning based techniques, so as to yield a sound and generic method to solve approximately a large class of multistage decision problems under uncertainty. The framework and solution techniques presented in the chapter are explained and illustrated on several examples. Along the way, we describe notions from decision theory that are relevant to sequential decision making under uncertainty in general.

1 INTRODUCTION

This chapter addresses decision problems under uncertainty for which complex decisions can be taken in several successive stages. By complex decisions, it is meant that the decisions are structured by numerous constraints or lie in high-dimensional spaces. Problems where this situation arises include capacity planning, production planning, transportation and logistics, financial management, and others (Wallace & Ziemba, 2005). While those applications are not currently mainstream domains of research in artificial intelligence, where many achievements have already been obtained for control problems with a finite number of actions and for problems where the uncertainty is reduced to some independent noise perturbing the dynamics of the controlled system, the interest for applications closer to operations research — where single instantaneous decisions may already be hard to find and where uncertainties from the environment may be delicate to model — and for applications closer to those addressed in decision theory — where complex objectives and potentially conflicting requirements have to be taken into account — seems to be growing in the machine learning community, as indicated by a series of advances in approximate dynamic programming motivated by such applications (Powell, 2007; Csáji & Monostori, 2008).

Computing strategies involving complex decisions calls for optimization techniques that can go beyond the simple enumeration and evaluation of the possible actions of an agent. Multistage stochastic programming, the approach presented in this chapter, relies on mathematical programming and probability theory. It has been recognized by several industries — mainly in energy (Kallrath, Pardalos, Rebennack, & Scheidt, 2009) and finance (Dempster, Pflug, & Mitra, 2008) — as a promising framework to formulate complex problems under uncertainty, exploit domain knowledge, use risk-averse objectives, incorporate probabilistic and dynamical aspects, while preserving structures compatible with large-scale optimization techniques.

Even for readers primarily concerned by robotics, the development of these techniques for sequential decision making under uncertainty is interesting to follow: Puterman (1994), citing Arrow (1958) on the early roots of sequential decision processes, recalls the role of the multi-period inventory models from the industry in the development of the theory of Markov Decision Processes (Chapter 3 of this book); we could also mention the role of applications in finance as a motivation for the early theory of multi-armed bandits and for the theory of sequential prediction, now an important field of research in machine learning.

The objective of the chapter is to provide a functional view of the concepts and methods proper to multistage stochastic programming.

To communicate the spirit of the approach, we use examples that are short in their description. We use the freely available optimization software cvx (Grant & Boyd, 2009), which has the merit of enabling occasional users of optimization techniques to conduct their own numerical experiments in Matlab (The MathWorks, Inc., 2004).

We cover our recent contributions on scenario tree selection and out-of-sample validation of optimized models (Defourny, Ernst, & Wehenkel, 2009), which suggest partial answers to issues concerning the selection of approximations/discretization of multistage stochastic programs, and to issues concerning the efficient comparison among those approximations.

Many details relative to optimization algorithms and specific problem classes have been left aside in our presentation. A more extensive coverage of these aspects can be found in J. Birge and Louveaux (1997); Shapiro, Dentcheva, and Ruszczyński (2009). These excellent sources also present many examples of formulations of stochastic programming models. Note, however, that there is no such thing as a general theory on multistage stochastic programming that would allow the many approximation/discretization schemes proposed in the technical literature (referenced later in the chapter) to be sorted out.

Background

Now, even if we insist on concepts, our presentation cannot totally escape from the fact that multistage stochastic programming uses optimization techniques from mathematical programming, and can harness advances in the field of optimization.

To describe what a mathematical program is, simply say that there is a function F, called the objective function, that assigns to $x \in X$ a real-valued cost $F(x)$, that there exists a subset C of X, called the feasibility set, describing admissible points x (by functional relations, not by enumeration), and that the program formulates our goal of computing the minimal value of F on C, written $\min_C F$, and a solution $x^* \in C$ attaining that optimal value, that is, $F(x^*) = \min_C F$. Note that the set of points $x \in C$ such that $F(x) = \min_C F$, called the optimal solution set and written $\operatorname{argmin}_C F$, need not be a singleton. Obviously, many problems can be formulated in that way, and what makes the interest of optimization theory is the clarification of the conditions on F and C that make a minimization problem well-posed ($\min_C F$ finite and attained) and efficiently solvable. For instance, in the chapter, we speak of convex optimization problems. To describe this class, imagine a point $\bar{x} \in C$, and assume that for each $x \in C$ in a neighborhood of $\bar{x}$, we have $F(x) \geq F(\bar{x})$. The class of convex problems is a class for which any such $\bar{x}$ belongs to the optimal solution set $\operatorname{argmin}_C F$. In particular, the minimization of F over C with F an affine function of $x \in \mathbb{R}^n$ (meaning that F has values $F(x) = \sum_{i=1}^n a_i x_i + a_0$) and $C = \{x : g_i(x) \leq 0,\ h_j(x) = 0 \text{ for } i \in I,\ j \in J\}$ for some index sets I, J and affine functions g_i, h_j, turns out to be a convex problem (linear programming) for which huge instances — in terms of the dimension of x and the cardinality of I, J — can be solved. Typically, formulation tricks are used to compensate the structural limitations on F, C by an augmentation of the dimension of x and the introduction of new constraints $g_i(x) \leq 0$, $h_j(x) = 0$. For example, minimizing the piecewise linear function $f(x) = \max\{a_i x + b_i : i \in I\}$ defined on $x \in \mathbb{R}$, with $I = \{1, \dots, m\}$ and $a_i, b_i \in \mathbb{R}$, is the same as minimizing the linear function $F(x,t) = t$ over the set $C = \{(x,t) \in \mathbb{R} \times \mathbb{R} : a_i x + b_i - t \leq 0,\ i \in I\}$. The trick can be particularized to $f(x) = |x| = \max\{x, -x\}$.

For more on applied convex optimization, we refer to Boyd and Vandenberghe (2004). But if the reader is ready to accept that conditions on F and C form well-characterized classes of problems, and that through some modeling effort one is often able to formulate interesting problems as an instance of one of those classes, then it is possible to elude a full description of those conditions.

Stochastic programming (Dantzig, 1955) is particular from the point of view of approximation and numerical optimization in that it involves a representation of the objective F by an integral (as soon as F stands for an expected cost under a continuous probability distribution), a large, possibly infinite number of dimensions for x, and a large, possibly infinite number of constraints for defining the feasibility set C. In practice, one has to work with approximations F', C', and be content with an approximate solution $x' \in \operatorname{argmin}_{C'} F'$. Multistage stochastic programming (the extension of stochastic programming to sequential decision making) is challenging in that small imbalances in the approximation can be amplified from stage to stage, and that x' may be lying in a space of dimension considerably smaller than the initial space for x. Special conditions might be required for ensuring the compatibility of an approximate solution x' with the initial requirement $x \in C$.

One of the main messages of the chapter is that it is actually possible to make use of supervised learning techniques (Hastie, Tibshirani, & Friedman, 2009) to lift x' to a full-dimensional approximate solution $\tilde{x} \in C$, and then use an estimate of the value $F(\tilde{x})$ as a feedback signal on the quality of the approximation of F, C by F', C'.

The computational efficiency of this lifting procedure based on supervised learning allows us to compare reliably many approximations of F and C, and therefore to sort out empirically, for a given problem or class of problems at hand, the candidate rules that are actually relevant for building better approximations. This new methodology for guiding the development of discretization procedures for multistage stochastic programming is exposed in the present chapter.

We recall that supervised learning aims at generalizing a finite set of examples of input-output pairs, sampled from an unknown but fixed distribution, to a mapping that predicts the output corresponding to a new input. This can be viewed as the problem of finding parameters maximizing the likelihood of the observed pairs (if the mapping has a parametric form); alternatively this can be viewed as the problem of selecting, from a hypothesis space of controlled complexity, the hypothesis that minimizes the expectation of the discrepancy between true outputs and predictions, measured by a certain loss function. The complexity of the hypothesis space is determined by comparing the performance of learned predictors on unseen input samples. Common methods for tackling the supervised learning problem include: neural networks (Hinton, Osindero, & Teh, 2006), support vector machines (Steinwart & Christman, 2008), Gaussian processes (Rasmussen & Williams, 2006), and decision trees (Breiman, Friedman, Stone, & Olshen, 1984; Geurts, Ernst, & Wehenkel, 2006). A method may be preferable to another depending on how easily one can incorporate prior knowledge in the learning algorithm, as this can make the difference between rich or poor generalization abilities.

Note that supervised learning is integrated to many approaches for tackling the reinforcement learning problem (Chapter 4 of this book). The literature on this aspect is large: see, for instance, Lagoudakis and Parr (2003); Ernst, Geurts, and Wehenkel (2005); Langford and Zadrozny (2005); Munos and Szepesvári (2008).

Organization of the Chapter

The chapter is organized as follows. We begin by presenting the multistage stochastic programming framework, the discretization techniques, and the considerations on numerical optimization methods that have an influence on the way problems are modeled. Then, we compare the approach to Markov Decision Processes, discuss the curse of dimensionality, and put in perspective simpler decision making models based on numerical optimization, such as two-stage stochastic programming with recourse or Model Predictive Control. Next, we explain the issues posed by the dominant approximation/discretization approach for solving multistage programs (which is suitable for handling both discrete and continuous random variables). A section is dedicated to the proposed extension of the multistage stochastic programming framework by techniques from machine learning. The proposal is followed by an extensive case study, showing how the proposed approach can be implemented in practice, and in particular how it allows to infer guidelines for building better approximations of a particular problem at hand. The chapter is complemented by a discussion of issues arising from the choice of certain objective functions that can lead to inconsistent sequences of decisions, in a sense that we will make precise. The conclusion indicates some avenues for future research.

2 THE DECISION MODEL

This section describes the multistage stochastic programming approach to sequential decision making under uncertainty, starting from elementary considerations. The reader may also want to take a look at the example at the end of this section (and to the few lines of Matlab code that implement it).

2.1 From Nominal Plans to Decision Processes

In their first attempt towards planning under uncertainty, decision makers often set up a course of actions, or *nominal plan* (reference plan), deemed to be robust to uncertainties in some sense, or to be a wise bet on future events. Then, they apply the decisions, often diverging from the nominal plan to better take account of actual events. To further improve the plan, decision makers are then led to consider (i) in which parts of the plan flexibility in the decisions may help to better fulfill the objectives, and (ii) whether the process by which they make themselves (or the system) "ready to react" impacts the initial decisions of the plan and the overall objectives. If the answer to (ii) is positive, then it becomes valuable to cast the decision problem as a sequential decision making problem, even if the net added value of doing so (benefits minus increased complexity) is unknown at this stage. During the planning process, the adaptations (or *recourse decisions*) that may be needed are clarified, their influence on prior decisions is quantified. The notion of nominal plan is replaced by the notion of *decision process*, defined as a course of actions driven by observable events. As distinct outcomes have usually antagonist effects on ideal prior decisions, it becomes crucial to determine which outcomes should be considered, and what importance weights should be put on these outcomes, in the perspective of selecting decisions under uncertainty that are not regretted too much after the dissipation of the uncertainty by the course of real-life events.

2.2 Incorporating Probabilistic Reasoning

In the *robust optimization* approach to decision making under uncertainty, decision makers are concerned by worst-case outcomes. Describing the uncertainty is then essentially reduced to drawing the frontier between events that should be considered and events that should be excluded from consideration. In that context, outcomes under consideration form the *uncertainty set*, and decision making becomes a game against some hostile opponent that selects the worst outcome from the uncertainty set. The reader will find in Ben-Tal, El Ghaoui, and Nemirovski (2009) arguments in favor of robust approaches.

In a *stochastic programming* approach, decision makers use a softer frontier between possible outcomes, by assigning weights to outcomes and optimizing some aggregated measure of performance that takes into account all these possible outcomes. In that context, the weights are often interpreted as a probability measure over the events, and a typical way of aggregating the events is to consider the expected performance under that probability measure.

Furthermore, interpreting weights as probabilities allows *reasoning under uncertainty.* Essentially, probability distributions are conditioned on observations, and Bayes' rule from probability theory (Chapter 2 of this book) quantifies how decision makers' initial beliefs about the likelihood of future events — be it from historical data or from bets — should be updated on the basis of new observations.

Technically, it turns out that the optimization of a decision process contingent to future events is more tractable (read: suitable to large-scale operations) when the "reasoning under uncertainty" part can be decoupled from the optimization process itself. Such a decoupling occurs in particular when the probability distributions describing future events are not influenced in any way by the decisions selected by the agent, that is, when the uncertainty is exogenous to the decision process. Examples of applications where the uncertainty can be treated as an exogenous process include capacity planning (especially in the gas and electricity industries), and asset and liability management. In both case, historical data allows to calibrate a model for the exogenous process.

2.3 The Elements of the General Decision Model

We are now in a position to describe the general decision model used throughout the chapter, and introduce some notations. The model is made of the following elements.

1. A sequence of random variables $\xi_1, \xi_2, \ldots, \xi_T$ defined on a probability space $(\Omega, \mathcal{B}, \mathbb{P})$. For a rigorous definition of the probability space, see e.g. Billingsley (1995). We simply recall that for a real-valued random variable ξ_t, interpreted, in the context of the rigorous definition of the probability space, as a $\mathcal{B}$-measurable mapping from Ω to $\mathbb{R}$ with values $\xi_t(\omega)$, the probability that $\xi_t \leq v$, written $\mathbb{P}\{\xi_t \leq v\}$, is the measure under $\mathbb{P}$ of the set $\{\omega \in \Omega : \xi_t(\omega) \leq v\} \in \mathcal{B}$. One can write $P(\xi_t \leq v)$ for $\mathbb{P}(\xi_t \leq v)$ when the measure $\mathbb{P}$ is clear from the context. If $\xi_1, \ldots, \xi_T$ are real-valued random variables, the function of $(v_1, \ldots, v_T)$ with values $\mathbb{P}\{\xi_1 \leq v_1, \ldots, \xi_T \leq v_T\}$ is the joint distribution function of $\xi_1, \ldots, \xi_T$. The smallest closed set Ξ in $\mathbb{R}^T$ such that $\mathbb{P}\{(\xi_1, \ldots, \xi_T) \in \Xi\} = 1$ is the support of measure $\mathbb{P}$ of $(\xi_1, \ldots, \xi_T)$, also called the support of the joint distribution. If the random variables are vector-valued, the joint distribution function can be defined by breaking the random variables into their scalar components. For simplicity, we may assume that the random variables have a joint density (with respect the Lebesgue measure for continuous random variables, or with respect to the counting measure for discrete random variables), written $\mathbb{P}(\xi_1, \ldots, \xi_T)$ by a slight abuse of notation, or $p(\xi_1, \ldots, \xi_T)$ if the measure $\mathbb{P}$ can be understood from the context. As several approximations to $\mathbb{P}$ are introduced in the sequel and compared to the exact measure $\mathbb{P}$, we always stress the appropriate probability measure in the notation.
The random variables represent the uncertainty in the decision problem, and their possible realizations (represented by the support of measure $\mathbb{P}$) are the possible observations to which the decision maker will react. The probability measure $\mathbb{P}$ serves to quantify the prior beliefs about the uncertainty. There is no restriction on the structure of the random variables; in particular, the random variables may be dependent. When the realization of $\xi_1, \ldots, \xi_{t-1}$ is known, there is a residual uncertainty represented by the random variables $\xi_t, \ldots, \xi_T$, the distribution of which in now conditioned on the realization of $\xi_1, \ldots, \xi_{t-1}$.

Table 1. Decision stages, setting the order of observations and decisions.

Stage	Available information for taking decisions			Decision
	Prior decisions	Observed outcomes	Residual uncertainty	
1	none	none	$\mathbb{P}(\xi_1, \ldots, \xi_T)$	u_1
2	u_1	ξ_1	$\mathbb{P}(\xi_2, \ldots, \xi_T \mid \xi_1)$	u_2
3	u_1, u_2	ξ_1, ξ_2	$\mathbb{P}(\xi_3, \ldots, \xi_T \mid \xi_1, \xi_2)$	u_3
$\vdots$				$\vdots$
T	$u_1, \ldots, u_{T-1}$	$\xi_1, \ldots, \xi_{T-1}$	$\mathbb{P}(\xi_T \mid \xi_1, \ldots, \xi_{T-1})$	u_T
optional: $T{+}1$	$u_1, \ldots, u_T$	$\xi_1, \ldots, \xi_T$	none	(u_{T+1})

For example, the evolution of the price of resources over a finite time horizon T can be represented, in a discrete-time model, by a random process $\xi_1, \ldots \xi_T$, with the dynamics of the process inferred from historical data.

2. A sequence of decisions u_1, u_2, …, u_T defining the decision process for the problem. Some models also use a decision u_{T+1}. We will assume that u_t is valued in a Euclidian space $\mathbb{R}^m$ (the space dimension m, corresponding to a number of scalar decisions, could vary with the index t, but we will not stress that in the notation).
   For example, a decision u_t could represent quantities of resources bought at time t.
3. A convention specifying when decisions should actually be taken and when the realizations of the random variables are actually revealed. This means that if ξ_{t-1} is observed before taking a decision u_t, we can actually adapt u_t to the realization of ξ_{t-1}. To this end, we identify *decision stages*: see Table 1. A row of the table is read as follows: at decision stage $t > 1$, the decisions $u_1, \ldots, u_{t-1}$ are already implemented (no modification is possible), the realization of the random variables $\xi_1, \ldots, \xi_{t-1}$ is known, the realization of the random variables $\xi_t, \ldots, \xi_T$ is still unknown but a density $\mathbb{P}(\xi_t, \ldots, \xi_T \mid \xi_1, \ldots, \xi_{t-1})$ conditioned on the realized value of $\xi_1, \ldots, \xi_{t-1}$ is available, and the current decision to take concerns the value of u_t. Once such a convention holds, we need not stress in the notation the difference between random variables ξ_t and their realized value, or decisions as functions of uncertain events and the actual value for these decisions: the correct interpretation is clear from the context of the current decision stage.
   The adaptation of a decision u_t to prior observations $\xi_1, \ldots, \xi_{t-1}$ will always be made in a deterministic fashion, in the sense that u_t is uniquely determined by the value of $(\xi_1, \ldots, \xi_{t-1})$.
   A sequential decision making problem has more than two decision stages inasmuch as the realizations of the random variables are not revealed simultaneously: the choice of the decisions taken between successive observations has to take into account some residual uncertainty on future observations. If the realization of several random variables is revealed before actually taking a decision, then the corresponding random variables should be merged into a single random vector; if several decisions are taken without intermediary ob-

servations, then the corresponding decisions should be merged into a single decision vector. This is how a problem concerning several time periods could actually be a two-stage stochastic program, involving two large decision vectors u_1 (first-stage decision, constant), u_2 (recourse decision, adapted to the observation of ξ_1). What is called a *decision* in a stochastic programming model may thus actually correspond to several actions implemented over a certain number of discrete time periods.

4. A sequence of feasibility sets $\mathcal{U}_1, \ldots, \mathcal{U}_T$ describing which decisions $u_1, \ldots, u_T$ are admissible. When $u_t \in \mathcal{U}_t$, one says that u_t is feasible. The feasibility sets $\mathcal{U}_2, \ldots, \mathcal{U}_T$ may depend, in a deterministic fashion, on available observations and prior decisions. Thus, following Table 1, $\mathcal{U}_t$ may depend on ξ_1, u_1, ξ_2, u_2, $\ldots$, ξ_{t-1} in a deterministic fashion. Note that prior decisions are uniquely determined by prior observations, but for convenience we keep track of prior decisions to parameterize the feasibility sets.
   An important role of the feasibility sets is to model how decisions are affected by prior decisions and prior events. In particular, a situation with no possible recourse decision ($\mathcal{U}_t$ empty at stage t, meaning that no feasible decision $u_t \in \mathcal{U}_t$ exists) is interpreted as a catastrophic situation to be avoided at any cost.
   We will always assume that the planning agent knows the set-valued mapping from the random variables $\xi_1, \ldots, \xi_{t-1}$ and the decisions $u_1, \ldots, u_{t-1}$ to the set $\mathcal{U}_t$ of feasible decisions u_t.
   We will also assume that the feasibility sets are such that a feasible sequence of decisions $u_1 \in \mathcal{U}_1, \ldots, u_T \in \mathcal{U}_T$ exists for all possible joint realizations of $\xi_1, \ldots, \xi_T$. In particular, the fixed set $\mathcal{U}_1$ must be nonempty. A feasibility set $\mathcal{U}_t$ parameterized only by variables in a subset of $\{\xi_1, \ldots, \xi_{t-1}\}$ must be nonempty for any possible joint realization of those variables. A feasibility set $\mathcal{U}_t$ also parameterized by variables in a subset of $\{u_1, \ldots, u_{t-1}\}$ must be implicitly taken into account in the definition of the prior feasibility sets, so as to prevent immediately a decision maker from taking a decision at some earlier stage that could lead to a situation at stage t with no possible recourse decision ($\mathcal{U}_t$ empty), be it for all possible joint realizations of the subset of $\{\xi_1, \ldots, \xi_{t-1}\}$ on which $\mathcal{U}_t$ depends, or for some possible joint realization only. These implicit requirements will affect in particular the definition of $\mathcal{U}_1$.
   For a technical example, interpret $a \geq b$ for any vectors $a, b \in \mathbb{R}^q$ as a shorthand for the componentwise inequalities $a_i \geq b_i$, $i = 1, \ldots, q$, assume that u_{t-1}, $u_t \in \mathbb{R}^m$, and take $\mathcal{U}_t = \{u_t \in \mathbb{R}^m : u_t \geq 0,\ A_{t-1}u_{t-1} + B_t u_t = h_t(\xi_{t-1})\}$ with A_{t-1}, $B_t \in \mathbb{R}^{q \times m}$ fixed matrices, and h_t an affine function of ξ_{t-1} with values in $\mathbb{R}^q$. If B_t is such that $\{B_t u_t : u_t \geq 0\} = \mathbb{R}^q$, meaning that for any $v \in \mathbb{R}^q$, there exists some $u_t \geq 0$ with $B_t u_t = v$, then this is true in particular for $v = h_t(\xi_{t-1}) - A_{t-1}u_{t-1}$, so that $\mathcal{U}_t$ is never empty. More details on this kind of sufficient conditions in the stochastic programming literature can be found in Wets (1974).
   One can use feasibility sets to represent, for instance, the dynamics of resource inflows and outflows, assumed to be known by the planning agent.
5. A performance measure, summarizing the overall objectives of the decision maker, that should be optimized. It is assumed that the decision maker knows the performance measure. In this chapter, we write the performance measure as the expectation of a function f that assigns some scalar value to

each realization of $\xi_1, \dots, \xi_T$ and $u_1, \dots, u_T$, assuming the integrability of f with respect to the joint distribution of $\xi_1, \dots, \xi_T$.
For example, one could take for f a sum of scalar products $\sum_{t=1}^{T} c_t \cdot u_t$, where c_1 is fixed and where c_t depends affinely on $\xi_1, \dots, \xi_{t-1}$. The function f would represent a sum of instantaneous costs over the planning horizon. The decision maker would be assumed to know the vector-valued mapping from the random variables $\xi_1, \dots, \xi_{t-1}$ to the vector c_t, for each t.
Besides the expectation, more sophisticated ways to aggregate the distribution of f into a single measure of performance have been investigated (Pflug & Römisch, 2007). An important element considered in the choice of the performance measure is the tractability of the resulting optimization problem.

The planning problem is then formalized as a mathematical programming problem. The formulation relies on a particular representation of the random process $\xi_1, \dots, \xi_T$ in connection with the decision stages, commonly referred to as the *scenario tree.*

2.4 The Notion of Scenario Tree

Let us call *scenario* an outcome of the random process $\xi_1, \dots, \xi_T$. A *scenario tree* is an explicit representation of the branching process induced by the gradual observation of $\xi_1, \dots, \xi_T$, under the assumption that the random variables have a discrete support. It is built as follows. A *root node* is associated to the first decision stage and to the initial absence of observations. To the root node are connected children nodes associated to stage 2, one child node for each possible outcome of the random variable ξ_1. Then, to each node of stage 2 are connected children nodes associated to stage 3, one for each outcome of ξ_2 given the observation of ξ_1 relative to the parent node. The branching process construction goes on until the last stage is reached; at this point, the outcomes associated to the nodes on the unique path from the root to a leaf define together a particular scenario, that can be associated to the leaf.

The probability distribution of the random variables is also taken into account. Probability masses are associated to the nodes of the scenario tree. The root node has probability 1, whereas children nodes are weighted by probabilities that represent the probability of the value to which they are associated, conditioned on the value associated to their ancestor node. Multiplying the probabilities of the nodes of the path from the root to a leaf gives the probability of a scenario.

Clearly, an exact construction of the scenario tree would require an infinite number of nodes if the support of $(\xi_1, \dots, \xi_T)$ is discrete but not finite. A random process involving continuous random variables cannot be represented as a scenario tree; nevertheless, the scenario tree construction turns out to be instrumental in the construction of approximations to nested continuous conditional distributions.

Branchings are essential to represent residual uncertainty beyond the first decision stage. At the planning time, the decision makers may contemplate as many hypothetical scenarios as desired, but when decisions are actually implemented, the decisions cannot depend on observations that are not yet available. We have seen that the decision model specifies, with decision stages, how the scenario

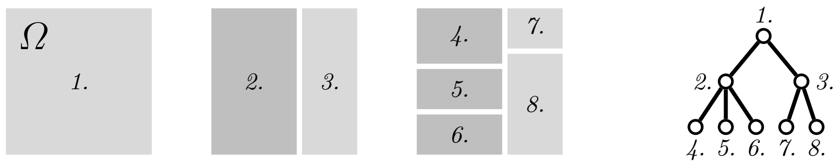

Figure 1. (From left to right) Nested partitioning of the event space Ω, starting from a trivial partition representing the absence of observations. (Rightmost) Scenario tree corresponding to the partitioning process.

actually realized will be gradually revealed. No branchings in the representation of the outcomes of the random process would mean that after conditioning on the observation of ξ_1, the outcome of $\xi_2, \ldots, \xi_T$ could be predicted (anticipated) exactly. Under such a representation, decisions spanning stages 2 to T would be optimized on the anticipated outcome. This would be equivalent to optimizing a nominal plan for $u_2, \ldots, u_T$ that fully bets on some scenario anticipated at stage 2.

To visualize how information on the realization of the random variables becomes gradually available, it is convenient to imagine nested partitions of the event space (Figure 1): refinements of the partitions appear gradually at each decision stage in correspondence with the possible realizations of the new observations. To each subregion induced by the partitioning of the event space can be associated a constant recourse decision, as if decisions were chosen according to a piecewise constant decision policy. On Figure 1, the surface of each subregion could also represent probabilities (then by convention the initial square has a unit surface and the thin space between subregions is for visual separation only). The dynamical evolution of the partitioning can be represented by a scenario tree: the nodes of the tree corresponds to the subregions of the event space, and the edges between subregions connect a parent subregion to its refined subregions obtained by one step of the recursive partitioning process.

Ideally, a scenario tree should cover the totality of possible outcomes of a random process. But unless the support of the distribution of the random variables is finite, no scenario tree with a finite number of nodes can represent exactly the random process and the probability measure, as we already mentioned, while even if the support is finite, the number of scenarios grows exponentially with the number of stages. How to exploit finite scenario tree approximations in order to extract good decision policies for general multistage stochastic programming problems involving continuous distributions will be extensively addressed in this chapter.

2.5 The Finite Scenario-Tree Based Approximation

In the general decision model, the agent is assumed to have access to the joint probability distributions, and is able to derive from it the conditional distributions listed in Table 1. In practice, computational limitations will restrict the quality of the representation of $\mathbb{P}$. Let us however reason at first at an abstract and ideal level to establish the program that an agent would solve for planning under uncertainty.

For brevity, let ξ denote $(\xi_1, \ldots, \xi_T)$, and let $\pi(\xi)$ denote a *decision policy* mapping realizations of ξ to realizations of the decision process $u_1, \ldots, u_T$. Let

$\pi_t(\xi)$ denote u_t viewed as a function of ξ. To be consistent with the decision stages, the policy must be *non-anticipative*, in the sense that u_t cannot depend on observations relative to subsequent stages. Equivalently one can say that π_1 must be a constant-valued function, π_2 a function of ξ_1, and in general π_t a function of $\xi_1, \ldots, \xi_{t-1}$ for $t = 2, \ldots, T$.

The planning problem can then be stated as the search for a non-anticipative policy π, restricted by the feasibility sets $\mathcal{U}_t$, that minimizes an expected total cost f spanning the decision stages and determined by the scenario ξ and the decisions $\pi(\xi)$:

$$\begin{aligned} \mathcal{S}: \quad & \text{minimize} \quad \mathbb{E}\{f(\xi, \pi(\xi))\} \\ & \text{subject to} \quad \pi_t(\xi) \in \mathcal{U}_t(\xi) \;\; \forall\, t, \\ & \quad \pi(\xi) \text{ non-anticipative} \;\; . \end{aligned}$$

Here we used an abstract notation which hides the nested expectations corresponding to the successive random variables, and the possible decomposition of the function f among the different decision stages indexed by t. It is possible to be even more general by replacing the expectation operator by a functional Φ assigning single numbers in $\mathbb{R} \cup \{\pm\infty\}$ to distributions. We also stressed the possible dependence of $\mathcal{U}_t$ on $\xi_1, u_1, \xi_2, u_2, \ldots, \xi_{t-1}$ by writing $\mathcal{U}_t(\xi)$.

A program more amenable to numerical optimization techniques is obtained by representing $\pi(\cdot)$ by a set of optimization variables for each possible argument of the function. That is, for each possible outcome $\xi^k = (\xi_1^k, \ldots, \xi_T^k)$ of ξ, one associates the optimization variables $(u_1^k, \ldots, u_T^k)$, written u^k for brevity. The non-anticipativity of the policy can be expressed by a set of equality constraints: for the first decision stage we require $u_1^k = u_1^j$ for all k, j, and for subsequent stages ($t > 1$) we require $u_t^k = u_t^j$ for each (k, j) such that $(\xi_1^k, \ldots, \xi_{t-1}^k) \equiv (\xi_1^j, \ldots, \xi_{t-1}^j)$.

A finite-dimensional approximation to the program $\mathcal{S}$ is obtained by considering a finite number n of outcomes, and assigning to each outcome a discrete probability p^k. This yields a formulation on a scenario tree covering the scenarios ξ^k:

$$\begin{aligned} \mathcal{S}': \quad & \text{minimize} \quad \textstyle\sum_{k=1}^n p^k \, f(\xi^k, u^k) \\ & \text{subject to} \quad u_t^k \in \mathcal{U}_t(\xi^k) \quad \forall\, k, \;\; \forall t \;\; ; \\ & \quad u_1^k = u_1^j \quad \forall\, k, j \;\; ; \\ & \quad u_t^k = u_t^j \quad \text{whenever} \\ & (\xi_1^k, \ldots, \xi_{t-1}^k) \equiv (\xi_1^j, \ldots, \xi_{t-1}^j) \;\; . \end{aligned}$$

Once again we used a simple notation ξ^k for designating outcomes of the process ξ, which hides the fact that outcomes can share some elements according to the branching structure of the scenario tree.

Non-anticipativity constraints can also be accounted for implicitly. A partial path from the root (depth 0) to some node of depth t of the scenario tree identifies some outcome $(\xi_1^k, \ldots, \xi_{t-1}^k)$ of $(\xi_1, \ldots, \xi_{t-1})$. To the node can be associated the decision u_{t+1}^k, but also all decisions u_{t+1}^j such that $(\xi_1^k, \ldots, \xi_t^k) \equiv (\xi_1^j, \ldots, \xi_t^j)$. Those decisions are redundant and can be merged into a single decision on the tree, associated to the considered node of depth t (the reader may refer to the scenario tree of Figure 1).

The finite scenario tree approximation is needed because numerical optimization methods cannot handle directly problems like $\mathcal{S}$, which cannot be specified with a finite number of optimization variables and constraints. The approximation may serve to provide an estimate of the optimal value of the original program; it may also serve to obtain an approximate first-stage decision u_1. Several aspects regarding the exploitation of scenario-tree approximations and the derivation of decisions for the subsequent stages will be further discussed in the chapter.

Finally, we point out that there are alternative numerical methods for solving stochastic programs (usually two-stage programs), based on the incorporation of the discretization procedure into the optimization algorithm itself, for instance by updating the discretization or carrying out importance sampling within the iterations of a given optimization algorithm (Norkin, Ermoliev, & Ruszczyński, 1998), or by using stochastic subgradient methods (Nemirovski, Juditsky, Lan, & Shapiro, 2009). Also, heuristics for finding policies directly have been suggested: a possible idea (akin to direct policy search procedures in Markov Decision Processes) is to optimize a combination of feasible non-anticipative basis policies $\pi^j(\xi)$ specified beforehand (Koivu & Pennanen, 2010).

2.6 Numerical Optimization of Stochastic Programs

The program formulated on the scenario tree is solved using numerical optimization techniques.

In principle, it is the size, the class and the structure of the program that determine which optimization algorithm is suitable. The size depends on the number of optimization variables and the number of constraints of the program. It is therefore influenced by the number of scenarios, the planning horizon, and the dimension of the decision vectors. The class of the program depends on the range of the optimization variables (set of permitted values), the nature of the objective function, and the nature of the equality and inequality constraints that define the feasibility sets. The structure depends on the number of stages, the nature of the coupling of decisions between stages, the way random variables intervene in the objective and the constraints, and on the joint distribution of the random variables (independence assumptions or finite support assumptions can sometimes be exploited). The structure determines whether the program can be decomposed in smaller parts, and where applicable, to which extent sparsity and factorization techniques from linear algebra can alleviate the complexity of matrix operations.

Note that the history of mathematical programming has shown a large gap between the complexity theory concerning some optimization algorithms, and the performance of these algorithms on problems with real-world data. A notable example (not directly related to stochastic programming, but striking enough to be mentioned here) is the traveling salesman problem (TSP). The TSP consists in finding a circuit of minimal cost for visiting n cities connected by roads (say that costs are proportional to road lengths). The dynamic programming approach, based on a recursive formulation of the problem, has the best known complexity bound: it is possible to find an optimal solution in time proportional to $n^2 2^n$. But in practice only small instances ($n \sim 20$) can be solved with the algorithm developed by Bellman (1962), due to the exponential growth of a list of optimal subpaths to consider. Linear programming approaches based on the simplex algorithm have an unattractive worst-case complexity, and yet such approaches have allowed to solve large instances of the problem — $n = 85900$ for the record on pla85900 obtained in 2006 — as explained in Applegate, Bixby, Chvátal, and Cook (2007).

In today's state of computer architectures and optimization technologies, multistage stochastic programs are considered numerically tractable, in the sense that numerical solutions of acceptable accuracy can be computed in acceptable time, when the formulation is convex. The covered class includes linear programs (LP) and convex quadratic programs (QP), which are similar to linear programs but have a quadratic component in their objective. A problem can be recognized to be convex if it can be written as a convex program; a program $\min_C F$ is convex if (i) the feasibility set C is convex, that is, $(1-\lambda)x + \lambda y \in C$ whenever $x, y \in C$ and $0 < \lambda < 1$ (Rockafellar, 1970, page 10); (ii) the objective function F is convex on C, that is, $F((1-\tau)x+\tau y) \leq (1-\tau)F(x)+\tau F(y)$, $0 < \tau < 1$, for every $x, y \in C$ (Rockafellar, 1970, Theorem 4.1). We refer to Nesterov (2003) for an introduction to complexity theory for convex optimization and to interior-point methods.

Integer programs (IP) and mixed-integer programs (MIP) are similar to linear programs but have integrality requirements on all (IP) or some (MIP) of their optimization variables. The research in stochastic programming for these classes is mainly focused on two-stage models: computationally-intensive methods for preprocessing programs so as to accelerate the repeated evaluation of integer recourse decisions (Schultz, Stougie, & Van der Vlerk, 1998); convex relaxations (Van der Vlerk, 2009); branch-and-cut strategies (Sen & Sherali, 2006). In large-scale applications, the modeling and numerical optimization aspects are closely integrated: see, for instance, the numerical study of Verweij, Ahmed, Kleywegt, Nemhauser, and Shapiro (2003). Solving multistage stochastic mixed-integer models is extremely challenging, but significant progress has been made recently (Escudero, 2009).

In our presentation, we focus on convex problems, and use Matlab for generating the data structure and values of the scenario trees, and cvx (Grant & Boyd, 2008, 2009) for formulating and solving the resulting programs — cvx is a modeling tool: it uses a language close to the mathematical formulation of the models, leading to codes that are slower to execute but less prone to errors.

2.7 Example

To fix ideas, we illustrate the scenario tree technique on a trajectory tracking problem under uncertainty with control penalization. In the proposed example, the uncertainty is such that the exact problem can be posed on a small finite scenario tree.

Say that a random process $\xi = (\xi_1, \xi_2, \xi_3)$, representing perturbations at time $t = 1, 2, 3$, has 7 possible outcomes, denoted by ξ^k, $1 \leq k \leq 7$, with known probabilities p^k:

k	1	2	3	4	5	6	7
ξ_1^k	-4	-4	-4	3	3	3	3
ξ_2^k	-3	2	2	-3	0	0	2
ξ_3^k	0	-2	1	0	-1	2	1
p^k	0.1	0.2	0.1	0.2	0.1	0.1	0.2

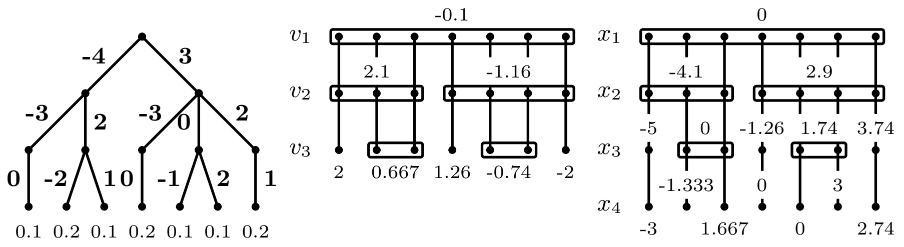

Figure 2. (Left) Scenario tree representing the 7 possible scenarios for a random process $\xi = (\xi_1, \xi_2, \xi_3)$. The outcomes ξ_t^k are written in bold, and the scenario probabilities p^k are reported at the leaf nodes. (Middle) Optimal actions v_t for the agent, displayed scenario per scenario, with frames around scenarios passing by a same tree node. (Right) Visited states x_t under the optimal actions, treated as artificial decisions (see text).

The random process is fully represented by the scenario tree of Figure 2 (Left) — the first possible outcome is $\xi^1 = (-4, -3, 0)$ and has probability $p^1 = 0.1$. Note that the random variables ξ_1, ξ_2, ξ_3 are not mutually independent.

Assume that an agent can choose actions $v_t \in \mathbb{R}$ at $t = 1, 2, 3$ (the notation v_t instead of u_t is justified in the sequel). The goal of the agent is the minimization of an expected sum of costs $\mathbb{E}\{\sum_{t=1}^3 c_t(v_t, x_{t+1}) \mid x_1 = 0\}$. Here $x_t \in \mathbb{R}$ is the state of a continuous-state, discrete-time dynamical system, that starts from the initial state $x_1 = 0$ and follows the state transition equation $x_{t+1} = x_t + v_t + \xi_t$. Costs $c_t(v_t, x_{t+1})$, associated to the decision v_t and the transition to the state x_{t+1}, are defined by $c_t = (d_{t+1} + v_t^2/4)$ with $d_{t+1} = |x_{t+1} - \alpha_{t+1}|$ and $\alpha_2 = 2.9$, $\alpha_3 = 0$, $\alpha_4 = 0$ (α_{t+1}: nominal trajectory, chosen arbitrarily; d_{t+1}: tracking error; $v_t^2/4$: penalization of control effort).

An optimal policy mapping observations $\xi_1, \dots, \xi_{t-1}$ to decisions v_t can be obtained by solving the following convex quadratic program over variables $v_t^k, x_{t+1}^k, d_{t+1}^k$, where k runs from 1 to 7 and t from 1 to 3, and over x_1^k trivially set to 0:

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^7 p^k \left[\sum_{t=1}^3 (d_{t+1}^k + (v_t^k)^2/4) \right] \\ \text{subject to} & -d_{t+1}^k \leq x_{t+1}^k - \alpha_{t+1} \leq d_{t+1}^k \quad \forall\, k, t \\ & x_1^k = 0 \ , \quad x_{t+1}^k = x_t^k + v_t^k + \xi_t^k \quad \forall\, k, t \\ & v_1^1 = v_1^2 = v_1^3 = v_1^4 = v_1^5 = v_1^6 = v_1^7 \\ & v_2^1 = v_2^2 = v_2^3 \ , \quad v_2^4 = v_2^5 = v_2^6 = v_2^7 \\ & v_3^2 = v_3^3 \ , \quad v_3^5 = v_3^6 \ . \end{array}$$

Here, the vector of optimization variables (v_1^k, x_1^k) plays the role of u_1^k, the vector (v_t^k, x_t^k, d_t^k) plays the role of u_t^k for $t = 2, 3$, and the vector (x_4^k, d_4^k) plays the role of u_4^k, showing that the decision process $u_1, \dots, u_{T+1}$ of the general multistage stochastic programming model can in fact include state variables and more generally any element that serves to evaluate costs conveniently.

The following code allows to formulate and solve the program using Matlab and cvx. It is almost a direct transcription of the program formulation, with variables and constraints defined in matrix form (column indices are relative to scenarios). Note that cvx replicates scalars if needed in componentwise constraints.

```
% problem data
xi    = [-4 -4 -4  3  3  3  3;...
         -3  2  2 -3  0  0  2;...
          0 -2  1  0 -1  2  1];
p     = [.1 .2 .1 .2 .1 .1 .2];
a     = [2.9 0 0]';
x1    = 0;  n  =  7;   T  =  3;
% call cvx toolbox
cvx_begin
 variables x(T+1,n) d(T,n) v(T,n)
 minimize( sum(sum(d*diag(p))) ...
    + sum(sum((v.^2)*diag(p)))/4);
 subject to
   -d <= x(2:T+1,:)-a(:,ones(1,n));
   x(2:T+1,:)-a(:,ones(1,n)) <=  d;
   x(1,:)     == x1;
   for t=1:T
     x(t+1,:) == ...
        x(t,:)+v(t,:)+xi(t,:);
   end
   v(1,2:n)   ==  v(1,1);
   v(2,2:3)   ==  v(2,1);
   v(2,5:n)   ==  v(2,4);
   v(3,3)     ==  v(3,2);
   v(3,6)     ==  v(3,5);
cvx_end
% display solution
cvx_optval, v, x
```

The code should return the optimal objective value $+7.3148$. The corresponding optimal solution is depicted on Figure 2. In the next section, we will discuss the differences between stochastic programming approaches and Markov Decision Processes. In this example, observe that the final solution can be recast as mappings $\tilde{\pi}_t$ from x_t to v_t, namely, $\tilde{\pi}_1(0) = -0.1$, $\tilde{\pi}_2(-4.1) = 2.1$, $\tilde{\pi}_2(2.9) = -1.16$, $\tilde{\pi}_3(-5) = 2$, $\tilde{\pi}_3(-1.26) = 1.26$, $\tilde{\pi}_3(0) = 0.667$, $\tilde{\pi}_3(1.74) = -0.74$, $\tilde{\pi}_3(3.74) = -2$. Hence in this case, the convenient assumption of an agent able to observe ξ_t instead of the system state x_t is not a fundamental restriction. Observe also that finding in this example the optimal mapping from x_t to v_t by a Markov Decision Process formulation is not straightforward, because the decision and state variables — to which the past states of the process ξ_t should be added, as the process is not memoryless — are continuous and unbounded.

3 COMPARISON TO RELATED APPROACHES

This section discusses several modeling and algorithmic complexity issues raised by the multistage stochastic programming framework and scenario-tree based decision making.

3.1 The Exogenous Nature of the Random Process

A frequent assumption made in the stochastic programming framework is that decision makers do not influence by their decisions the realization of the random process representing the uncertainty. The random process is said to be *exogenous*. This allows to simulate, select and organize in advance possible realizations of the exogenous process, before any observation is actually made, and then optimize jointly (by opposition to individually for each scenario) the decisions contingent to the possible realizations.

The need to decouple the description of uncertainties and the optimization of decisions might appear at first as a strong limitation on the situations that can be modeled and treated by stochastic programming techniques. This impression is in part justified for a large family of problems of control theory in which the uncertainty is identified to some zero-mean noise perturbing the observations or the dynamics of the system, or when the uncertainty is understood as the uncertainty on the value of system parameters However, in another large family of sequential decision making problems under uncertainty, major sources of uncertainty are precisely the ones that are the less influenced by the behavior of the decision makers (weather, interest rate evolution, accidental pollution, new regulations, for example). We also note that random processes strongly influenced by the behavior of the decision makers can sometimes be handled by incorporating them to the initial decision process and treating them as a virtual decision process.

A probabilistic reasoning based on a subset of possible scenarios could easily be tricked by an adversarial random process that would exploit one of the scenarios discarded during the planning process. In many practical problems however, the environment is not totally adversarial. In situations where the environment is mildly adversarial, it is often possible to choose measures of performances that are more robust to bad outcomes, and that can still be optimized in a tractable way. We will come back to issues posed by risk-sensitive models for sequential decision making at the end of the chapter.

Finally, it is easier in terms of sample complexity to learn a model (find model parameters from finite data sets) for an exogenous process than for an endogenous process. Learning a model for an exogenous process is possible from observations of the process, such as time series, whereas learning a model for an endogenous process forces us to be able to simulate possible state transitions for every possible action, or at least to have at one's disposal a fairly exhaustive data set relating actions to state transitions.

The scheduling of electric power units (Carpentier, Cohen, & Culioli, 1996; Sen, Yu, & Genc, 2006) and the management of cash flows, assets and liabilities (Dempster et al., 2008) are example of sequential decision making problems with exogenous processes following sophisticated models.

3.2 Comparison to Markov Decision Processes

In Markov Decision Processes (MDP), the decision maker seeks to optimize a performance criterion decomposed into a sum of instantaneous rewards. The information state of the decision maker at time t coincides with the state x_t of a dynamical system For simplicity, we do not consider in this discussion partial observability (POMDP) or risk-sensitivity, for which the system state need not be the information state of the agent. Optimal decision policies are often found by a reasoning based on the dynamic programming principle, to which is essential the notion of state as a sufficient statistic for representing the complete history of the system's evolution and agent's beliefs.

Multistage stochastic programming problems could be viewed as a subclass of finite-horizon Markov Decision Processes, by identifying the growing history of observations $(\xi_1, \ldots, \xi_{t-1})$ to the agent's state. However, the mathematical assumptions under the MDP and the stochastic programming formulations are in fact quite different. Complexity results suggest that the algorithmic resolution of MDPs is efficient when the decision space is finite and small (Littman, Dean, & Kaelbling, 1995; Kearns, Mansour, & Ng, 2002), while for the scenario-tree based stochastic programming framework, the resolution is efficient when the optimization problem is convex — in particular the decision space is continuous — and the number of decision stages is small (Shapiro, 2006).

One of the main appeals of stochastic programming techniques is their ability to deal efficiently with high-dimensional continuous decision spaces structured by numerous constraints, and with sophisticated, non-memoryless random processes. At the same time, if stochastic programming models have traditionally been used for optimizing long-term decisions that are implemented once and have lasting consequences, for example in network capacity planning, they are now increasingly used in the context of near-optimal control strategies that Bertsekas (2005) calls *limited-lookahead* strategies. In this usage, at each decision stage an updated model over the remaining planning horizon is rebuilt and optimized on the fly, from which only the first-stage decisions are actually implemented. Indeed, when a stochastic program is solved on a scenario tree, the initial search for a decision policy degenerates into the search for sequences of decisions relative to the scenarios covered by the tree. The first-stage decision does not depend on observations and can thus always be implemented on any new scenario, whereas the recourse decisions relative to any particular scenario in the tree could be infeasible on a new scenario, especially if the feasibility sets depend on the random process.

3.3 The Curse of Dimensionality

The *curse of dimensionality* is an algorithmic-complexity phenomenon by which computing optimal policies on higher dimensional input spaces requires an exponential growth of computational resources, leading to intractable problem formulations. In dynamic programming, the input space is the state space or a reduced parametrization of it. In practice the curse of dimensionality limits attempts to cover inputs to spaces embedded in $\mathbb{R}^d$ with d at most equal to 5-10.

Approximate Dynamic Programming methods (ADP) (Bertsekas, 2005) and Reinforcement Learning approaches (RL) (Sutton & Barto, 1998) help to mitigate the curse of dimensionality, for instance by attempting to cover only the regions of the input space that are actually reached under an optimal policy. An exploratory component may be added to the original dynamic programming solution strategy so as to discover the interesting regions of the input space by testing decisions. Those approaches work well in several cases:

– The structure of a near-optimal policy is known a priori. For example, policy search methods work well when a near-optimal policy can be described by a small number of parameters. Value-function based methods work well when there is a finite and rather small set of actions, known a priori, that are the elementary building blocks of a near-optimal policy, and are used to drive the exploratory phase. Such situations are often exploited in robotics. For instance, the fundamental building blocks of near-optimal policies can be reduced to a limited number of motor primitives optimized separately.

– The structure of the optimization problem is such that the promising decisions and input space regions identifiable early in the exploratory phase correspond to those that are actually relevant for a near-optimal policy. This ensures the practical success of optimistic exploratory strategies, that refine decisions within regions identified as promising. This situation typically arises in problems where the stochastic part comes from a noise process that slightly disturbs the dynamics of the system.

Stochastic programming algorithms do not rely on the covering of the state space of dynamic programming. Instead, they rely on the covering of the random exogenous process, which needs not correspond to the complete state space (see how the auxiliary state x_t is treated in the example of the previous section). The complement of the state space and the decision space are "explored" during the optimization procedure itself. The success of the approach will thus depend on the tractability of the joint optimization in those spaces, and not on insights on the structure of near-optimal policies.

In multistage stochastic programming approaches, the curse of dimensionality is present when the number of decision stages increases, and in face of high-dimensional exogenous processes. Therefore, methods that one could call, by analogy to ADP, *approximate stochastic programming methods*, will attempt to cover only the realizations of the exogenous random process that are truly needed to obtain near-optimal decisions. These methods work with a number of scenarios that does not grow exponentially with the dimension of the exogenous process and the number of stages.

3.4 The Value of Multistage Stochastic Programming

Due to the curse of dimensionality, multistage stochastic programming is in competition with more tractable decision models. At the same time it provides a unifying framework between several simplified decision making paradigms, that we now describe.

Reduction to Model Predictive Control. A radical simplification consists in discarding the detailed probabilistic information on the uncertainty, taking a nominal scenario, and optimizing decisions on the nominal scenario. As the common practice for defining a nominal scenario is to replace random variables by their expectation, the resulting problem on the nominal scenario is called the *expected value problem*, the solution of which constitutes a nominal plan. Even if the nominal plan could be used as an *open-loop decision policy*, that is, implemented over the complete planning horizon, decision makers will usually want to recompute the plan at the next decision stage by solving an updated expected value problem on a new nominal scenario that incorporates the observations. In the control community, the approach is known as Model Predictive

Control (MPC). We refer the reader with a background in reinforcement learning to Ernst, Glavic, Capitanescu, and Wehenkel (2009) for discussions on this area of research.

An indicator of the value of multistage programming decisions over model predictive control decisions is given by the *value of the stochastic solution* (VSS). To make the notion precise, let us define successively:

– V^*, the optimal value of the multistage stochastic program $\min_\pi \mathbb{E}\{f(\xi, \pi(\xi))\}$. For notational simplicity, we adopt the convention that $f(\xi, \pi(\xi)) = \infty$ if the policy π is anticipative or yields infeasible decisions.
– $\zeta = (\zeta_1, \ldots, \zeta_T)$, the nominal scenario.
– u^ζ, the optimal solution to the expected value problem $\min_u f(\zeta, u)$. Note that the optimization is over a single fixed sequence of feasible decisions; the problem data is determined by ζ.
– u_1^ζ, the first-stage decision of u^ζ.
– V^ζ, the optimal value of the multistage stochastic program $\min_\pi \mathbb{E}\{f(\xi, \pi(\xi))\}$ subject to the additional constraint $\pi_1(\xi) = u_1^\zeta$ for all ξ. If by a slight abuse of notation, we write π_1, viewed as an optimization variable, for the value of the constant-valued function π_1, then the additional constraint is simply $\pi_1 = u_1^\zeta$. By definition, V^ζ is the value of a policy implementing the first decision from the expected value problem, and then selecting optimal recourse decisions for the subsequent decision stages. The recourse decisions differ in general from those that would be selected by a policy optimal for the original multistage program.

The VSS is then defined as the difference $V^\zeta - V^* \geq 0$. For maximization problems, it would be defined by $V^* - V^\zeta \geq 0$. J. Birge and Louveaux (1997) describe special cases (with two decision stages, and restrictions on the way randomness affects problem data) for which it is possible to compute bounds on the VSS. They also come to the conclusion, from their survey of works studying the VSS, that there is no rule that can predict a priori whether the VSS is low or high for a given problem instance — for example increasing the variance of random variables may increase or decrease the VSS.

Reduction to Two-Stage Stochastic Programming. A less radical simplification consists in discarding the distinction between recourse stages, keeping in the model a first stage (associated to full uncertainty) and a second stage (associated to the full knowledge of the realized scenario). A multistage model degenerates into a two-stage model when the scenario tree has branchings only at one stage (we have already described how random variables and decisions should be merged if observations and decisions are not intertwined). The situation arises for instance when scenarios are sampled over the full horizon independently: the tree has then branchings only at the root. Huang and Ahmed (2009) define the *value of multistage programming* (VMS) as the difference of the optimal values of the multistage model versus the two-stage model. The authors establish bounds on the VMS and describe an application (in the semiconductor industry) where the VMS is high. Note however that a generalization of the notion of VSS would rather quantify how multistage decisions outperform two-stage decisions when those two-stage decisions are implemented with model rebuilding at each stage, in the manner of the Model Predictive Control scheme.

Reduction to Heuristics based on Parametric Optimization. As an intermediate simplification between the expected value problem and the reduction to a two-stage model, it is possible to optimize sequences of decisions separately on each scenario. The decision maker can then use some averaging, consensus or selection strategy to implement a first-stage decision inferred from the so-obtained ensemble of first-stage decisions. Here again, the model should be rebuilt with updated scenarios at each decision stage.

The problem of computing optimal decisions separately for each scenario is known as the *distribution problem*. The problem appears in the definition of the *expected value of perfect information* (EVPI), which quantifies the additional value that a decision maker could reach in expectation if he or she were able to predict the future. To make the notion precise, let V^* denote as before the optimal value of the multistage stochastic program $\min_\pi \mathbb{E}\{f(\xi, \pi(\xi))\}$ over non-anticipative policies π; let $V(\xi)$ denote the optimal value of the deterministic program $\min_u f(\xi, u)$; and let V^A be the expected value of $V(\xi)$, according to the distribution of ξ. Observe that V^A is also the optimal value of the program $\min_{\pi^A} \mathbb{E}\{f(\xi, \pi^A(\xi))\}$ over anticipative policies π^A, the optimization of which is now decomposable among scenario subproblems. The EVPI is then defined as the difference $V^* - V^A \geq 0$. For maximization problems, it is defined by $V^A - V^* \geq 0$. Intuitively, the EVPI is high when having to delay adaptations to final outcomes due to a lack of information results in high costs.

The EVPI is usually interpreted as the price a decision maker would be ready to pay to know the future (Raiffa & Schlaifer, 1961; J. R. Birge, 1992). The EVPI also indicates how valuable the dependence of decision sequences is on the particular scenario they are optimized over. Mercier and Van Hentenryck (2007) show on an example with low EVPI how a strategy based on a particular aggregation of decisions optimized separately on deterministic scenarios can be arbitrarily bad. Thus even if the EVPI is low, heuristics based on the decisions of anticipative policies can perform poorly.

This does not mean that the approach cannot perform well in practice. Van Hentenryck and Bent (2006) have studied and refined various aggregation and regret-minimization strategies on a series of stochastic combinatorial problems already hard to solve on a single scenario, as well as schemes that build a bank of pre-computed reference solutions and then adapt them online to accelerate the optimization on new scenarios. They show that their strategies perform well on vehicle routing applications.

Remark on the Progressive Hedging Algorithm. The progressive hedging algorithm (PHA) of Rockafellar and Wets (1991) is a decomposition method that computes the solution to a multistage stochastic program on a scenario tree by solving repeatedly separate subproblems on the scenarios covered by the tree. First-stage decisions and other decisions coupled by non-anticipativity constraints are obtained by aggregating the decisions of the concerned scenarios, in the spirit of the heuristics based on the distribution problem presented above. The algorithm modifies the scenario subproblems at each iteration to make the decisions coupled by non-anticipativity constraints converge towards a common and optimal decision.

As the iterations are carried out, first-stage decisions evolve from decisions hedged by the aggregation strategy to decisions hedged by the multiple recourse decisions computed on the scenario tree. Therefore, the progressive hedging algorithm shows that there can be a smooth conceptual transition between the decision model based on the distribution problem and the decision model based on the multistage stochastic programming problem.

3.5 Example

We illustrate the computation of the VSS and the EVPI on an artificial multistage problem, with numerical parameters chosen in such a way that the full multistage model is valuable. By valuable we mean that the presented simplified decision-making schemes will output first-stage decisions that are suboptimal. If those decisions were implemented, and subsequently the best possible recourse decisions were applied, the value of the objective over the full horizon would be significantly suboptimal.

Let w_1, w_2, w_3 be mutually independent random variables uniformly distributed on $\{+1, -1\}$. Let $\xi = (\xi_1, \xi_2, \xi_3)$ be a random walk such that $\xi_1 = w_1$, $\xi_2 = w_1 + w_2$, $\xi_3 = w_1 + w_2 + w_3$. Let the 8 equiprobable outcomes of ξ form a scenario tree and induce non-anticipativity constraints (the tree is a binary tree of depth 3). Consider the decision process $u = (u_1, u_2, u_3)$ with $u_2 \in \mathbb{R}$ and $u_t = (u_{t1}, u_{t2}) \in \mathbb{R}^2$ for $t = 1, 3$. Then consider the multistage stochastic program

$$\begin{aligned}
&\text{maximize} \\
&\quad \tfrac{1}{8}\textstyle\sum_{k=1}^{8}\{[0.8u_{11}^k - 0.4(u_2^k/2 + u_{31}^k - \xi_3^k)^2] \\
&\qquad + u_{32}^k\xi_3^k + [1 - u_{11}^k - u_{12}^k]\} \\
&\text{subject to} \\
&\quad u_{11}^k + u_{12}^k \le 1 \quad \forall k \\
&\quad -u_{11}^k \le u_2^k \le u_{11}^k \quad \forall k \\
&\quad -u_{1j}^k \le u_{3j}^k \le u_{1j}^k \quad \forall k \text{ and } j = 1, 2 \\
&\quad \text{C1: } u_1^k = u_1^1 \quad \forall k \\
&\quad \text{C2: } u_2^k = u_2^{k+1} = u_2^{k+2} = u_2^{k+3} \qquad \text{for } k = 1, 5 \\
&\quad \text{C3: } u_3^k = u_3^{k+1} \qquad \text{for } k = 1, 3, 5, 7 \ .
\end{aligned}$$

The non-anticipativity constraints C1, C2, C3, which are convenient to state the problem, indicate in practice the redundant optimization variables that can be eliminated.

- The optimal value of the multistage stochastic program is $V^* = 0.35$ with optimal first-stage decision $u_1^* = (1, 0)$.
- The *expected value problem* for the mean scenario $\zeta = (0, 0, 0)$ is obtained by setting momentarily $\xi^k = \zeta \ \forall \ k$ and adding the constraints C2': $u_2^k = u_2^1 \ \forall k$ and C2': $u_3^k = u_3^1 \ \forall k$. Its optimal value is 1 with first-stage decision $u_1^\zeta = (0, 0)$. When equality constraints are made implicit the problem can be formulated using 5 scalar optimization variables only.
- The *two-stage relaxation* is obtained by relaxing the constraints C2, C3. Its optimal value is 0.6361 with $u_1^k := u_1^{\text{II}} = (0.6111, 0.3889)$.

– The *distribution problem* is obtained by relaxing the constraints C1, C2, C3. Its optimal value is $V^A = 0.6444$. The two extreme scenarios $\xi^1 = (1, 2, 3)$ and $\xi^8 = (-1, -2, -3)$ have first-stage decisions $u_1^1 = u_1^8 = (0.7778, 0.2222)$ and value -0.0556. The 6 other scenarios have $u_1^k = (0.5556, 0.3578)$ and value 0.8778, $k = 2, \dots, 7$. Note that in general, (i) scenarios with the same optimal first-stage decision and values may still have different recourse decisions, and (ii) the first-stage decisions can be distinct for all scenarios.
– The EVPI is equal to $V^A - V^* = 0.2944$.
– Solving the multistage stochastic program with the additional constraint $\mathrm{C1}^\zeta : u_1^k = u_1^\zeta \; \forall k$ yields an upper bound on the optimal value of any scheme using the first-stage decision of the expected value problem. This value is $V^\zeta = -0.2$.
– The VSS is equal to $V^* - V^\zeta = 0.55$.
– Solving the multistage stochastic program with the additional constraint $\mathrm{C1}^{\mathrm{II}}$: $u_1^k = u_1^{\mathrm{II}} \; \forall k$ yields an upper bound on the optimal value of any scheme using the first-stage decision of the two-stage relaxation model. This value is $V^{\mathrm{II}} = 0.2431$. Thus, the value of the multistage model over a two-stage model, in our sense (distinct from the VMS of Huang and Ahmed (2009)), is at least $V^* - V^{\mathrm{II}}$=0.1069.

To summarize, observe the collapse of the optimal value from $V^* = 0.35$ to $V^{\mathrm{II}} = 0.2431$ (with the first-stage decision of the two-stage model) and then to $V^\zeta = -0.2$ (with the first-stage decision of the expected value model).

We can also consider the anticipative decision sequences of the distribution problem, and check if there exists plausible strategies that could exploit the set of first-stage decisions to output a good first-stage decision (with respect to any decision-making scheme for the subsequent stages).

– Selection strategy: Solving the multistage stochastic program with a constraint that enforces one of the first-stage decisions extracted from the distribution problem yields the following results: optimal value 0.3056 if $u_1^k = (0.7778, 0.2222)$, optimal value 0.2167 if $u_1^k = (0.5556, 0.3578)$. But one has to concede that in contrast to other simplified models, for which we solve multistage programs only to measure the quality of a suboptimal first-stage decision, the selection strategy needs good estimates of the different optimal values to actually output the best decision.
– Consensus strategy: The outcome of a majority vote out of the set of the 8 first-stage decisions would be the decision $(0.5556, 0.3578)$ associated to the scenarios 2 to 7. With value 0.2167, this turns out to be the worst decision between $(0.7778, 0.2222)$ and $(0.5556, 0.3578)$.
– Averaging strategy: The mean first-stage decision of the set of 8 first-stage decisions is $\bar{u}_1 = (0.6111, 0.3239)$. Solving the multistage program with $u_1^k = \bar{u}_1$ for all k yields the optimal value 0.2431.

The best result is the value 0.3056 obtained by the selection strategy. Note that we are here in a situation where the multistage program and its variants could be solved exactly, that is, with a scenario tree representing the possible outcomes of the random process exactly.

4 PRACTICAL SCENARIO-TREE APPROACHES

We now focus on a practical question essential to the deployment of a multistage stochastic programming model: if a problem has to be approximately represented by a scenario tree in order to compute a decision strategy, how should a tractable and at the same time representative scenario-tree approximation be selected for a given problem?

After some background on discretization methods for two-stage stochastic programming, we pose the scenario tree building problem in an abstract way and then discuss the antagonist requirements that make its solution very challenging. Then we review the main families of methods proposed in the literature to build tractable scenario-tree approximations for a given problem, and highlight their main properties from a theoretical point of view.

Given the difficulty of determining a priori good scenario-tree approximations for many problems of practical interest (a difficulty which is to some extent surprising, given the practical success of related approximation methods for two-stage stochastic programming), there is a growing consensus on the necessity of being able to test a posteriori the quality of scenario-tree based approximations on a large independent sample of new scenarios. We present in this light a standard strategy based on the so-called shrinking-horizon approach — the term is used, for instance, in Balasubramanian and Grossmann (2003).

4.1 Approximation Methods in Two-stage Stochastic Programming

Let $\mathcal{S}$ denote a two-stage stochastic program, where the uncertainty is modeled by a random vector ξ, possibly of high-dimension, following a certain distribution with either a discrete support of large cardinality, or a continuous support. Let $\mathcal{S}'$ be an approximation to $\mathcal{S}$, where ξ is approximated by a random vector ξ' that follows a distribution with a finite discrete support, the cardinality of the support being limited by the fact that to each possible realization of ξ' is associated optimization variables for representing the corresponding recourse decisions. To obtain a good approximation, one would ideally target the problem of finding a finite discrete distribution for ξ' (values for the support and associated probability masses) such that any first-stage decision u_1' optimal for $\mathcal{S}'$ yields on $\mathcal{S}$ a minimal regret, in the sense that with optimal recourse decisions, the value on $\mathcal{S}$ of the solution made of u_1' and of optimal recourse decisions is close to the exact optimal value of $\mathcal{S}$. By analogy to the VSS, we could also say that the distribution for ξ' should minimize the value of the exact program $\mathcal{S}$ with respect to the approximate program $\mathcal{S}'$.

Many authors have found it more convenient to restrict the attention on the problem of finding a finite discrete distribution for ξ' such that the optimal value of $\mathcal{S}'$ is close to the optimal value of $\mathcal{S}$, and the solutions u_1' optimal for $\mathcal{S}'$ are close to solutions optimal for $\mathcal{S}$. For this approach to work, one might want to impose some weak form of continuity of the objective of $\mathcal{S}$ with respect to solutions. One may also want to ensure that small perturbations of the probability measure for ξ have a bounded effect on the perturbation of optimal solutions u_1.

An interesting deterministic approach (Rachev & Römisch, 2002) consists in analyzing the structure of optimal policies for a given problem class, the structure of the objective when the optimal policy is implicitly taken into account, and inferring from it a relevant measure of distance between two distributions, based on worst-case differences among objectives in a class of functions having the identified structure (or in a larger class if this is technically convenient for the computation of the distance measure). Finding a good approximation to a two-stage stochastic program is then reformulated as the problem of finding a discrete distribution minimizing the relevant probability distance to the original distribution. Note that probability distance minimization problems can be difficult to solve, especially on high-dimensional distributions. Thus, the approach, which reformulates the ideal approximation problem as the optimal quantization of the initial probability distribution, can have essentially two sources of suboptimality with respect to the ideal approximation problem: (i) the class of functions over which worst-case distances are evaluated are as large as needed for the tractable computation of the distance; (ii) the minimal distance between probability measures is not necessarily attained in practice. Despite these limitations, the approach has been shown to work well. Moreover, the reduction of the initial approximation problem to an optimal quantization problem indicates the relevance of existing work on vector quantization and probability density estimation (MacKay, 2003, Chapter 20), and on discretization methods explored in approximate dynamic programming.

Randomized approaches are based on Monte Carlo sampling and its many extensions, including variance reduction techniques, and quasi Monte Carlo techniques. All these techniques have more or less been tried for solving two-stage stochastic programs: Infanger (1992), for instance, investigates importance sampling. They have been shown to work well in practice. Random approximations based on Monte Carlo have been shown to be consistent, in the sense that with an infinite number of samples, optimal solutions to discretized programs can converge to solutions optimal for $\mathcal{S}$. More detailed results can be found in Shapiro (2003b).

4.2 Challenges in the Generation of Scenario Trees

In two-stage stochastic programming, the large or infinite set of recourse decisions of the original program is reduced to a finite set of recourse decisions for the approximation. Hence the exact and approximate solutions lie in different spaces and cannot be compared directly. Still, recourse decisions can be treated implicitly, as if they were already incorporated to the objective function, and as if the only remaining element to optimize were the first-stage decision.

In multistage stochastic programming, we face the same issue: we cannot compare solutions directly. But now, treating all recourse decisions implicitly leads to a dilution of exploitable structural properties of the objective. The classes of functions respecting those weak properties are larger. Worst-case distances between functions in such classes may cease to guide satisfactorily a discretization procedure. In addition, discretization problems are posed over larger spaces, making them more difficult to solve, even approximately.

For these reasons, rather than presenting the generation of scenario trees as a natural extension of discretization methods for two-stage stochastic programming, with the incorporation of branchings for representing the nested conditional probability densities, we state the problem in a more open way, which also highlights complexity aspects:

Construct a tractable algorithm $\mathcal{A}$ that

- *given a multistage stochastic program* $\mathcal{S} : \min_\pi \mathbb{E}\{f(\xi, \pi(\xi))\}$ *defined over a probability space* $(\Omega, \mathcal{B}, \mathbb{P})$ *with objective* f *(including by convention the constraints) and non-anticipative policies* $\pi(\xi)$*,*
- *will produce an approximate finite-dimensional surrogate program of the form* $\mathcal{S}' : \min_u \sum_{k=1}^{n} p^k\{g(\xi^k, u^k)\}$*, defined over some reduced space* $(\Omega', \mathcal{B}', \mathbb{P}')$ *and objective* g*, and from which a surrogate policy* $\hat{\pi}(\xi)$ *subject to non-anticipativity constraints may be computed in a tractable way,*
- *with the goal of making the regret*

$$\mathcal{R} := \mathbb{E}\{f(\xi, \hat{\pi}(\xi))\} - \min_\pi \mathbb{E}\{f(\xi, \pi(\xi))\} \geq 0$$

as small as possible.

Notice that we allow, for the sake of generality, that the surrogate program may refer to a function g different from the original objective f, and that we impose that the algorithm $\mathcal{A}$, the solving strategy associated to the problem $\mathcal{S}'$, as well as the evaluation of the induced policy $\hat{\pi}$ on any new scenario, are all tractable. At this stage, we do not specify how $\hat{\pi}$ is inferred or understood; $\hat{\pi}$ needs to be introduced here only to be able to write a valid expression for the regret on the original multistage program.

Depending on situations, the problem $\mathcal{S}$ (random process model and function f) can be described analytically, or be only accessible through sampling and/or simulation. The problem $\mathcal{S}'$ will be described by a scenario tree and the choice of the function g, under limitations intrinsically due to the tractability of the optimization of the approximate program.

As we have seen, there are many derived decision-making schemes and usages of the multistage stochastic programming framework. Also, various classes of optimization programs can be distinguished — with the main distinctions being between two-stage and multistage settings, and among linear, convex, and integer/mixed-integer formulations — and thus several possible families of functions over which one might attempt to minimize a worst-case regret.

In the stochastic programming literature, several scenario tree generation strategies have been studied. The scenario tree generation problem is there often viewed in one or another of two reduced ways with respect to the above definition, namely

(i) as the problem of finding a scenario tree with an associated optimal value $\min_u \sum_{k=1}^{n} p^k\{f(\xi^k, u^k)\}$ close to the exact optimal value $\min_\pi \mathbb{E}\{f(\xi, \pi(\xi))\}$, or
(ii) as the problem of finding a scenario tree with its associated optimal first-stage decision $\hat{u}_1$ close to a first-stage decision π_1 optimal for the exact program.

Indeed, version (i) is useful when the goal is merely to estimate the optimal value of the original program $\mathcal{S}$, while version (ii) is useful when the goal is to extract only the first stage decision, assuming that later on, recourse decisions are recomputed using a similar algorithm, given the new observations.

The generic approximation problem that we have described is more general, since it covers also the case where the scenario tree approach may be exploited offline to extract a complete policy $\hat{\pi}(\xi)$ that may then be used later on, in a stand-alone fashion for decision making over arbitrary scenarios and decision steps, be it in the real world or in the context of Monte Carlo simulations.

To give an idea of theoretical results established in the scenario tree generation literature, we now briefly discuss two representative trends of research: work that study Monte Carlo methods for building the tree, and work that seek to minimize in a deterministic fashion a certain measure of discrepancy between the original process and the approximate process represented by the scenario tree.

Monte Carlo Scenario Tree Sampling Methods. Monte Carlo methods have several advantages: they are easy to implement and they scale well with the dimension, in the sense that with enough samples, one can get close to the statistical properties of high-dimensional target distributions with high probability. The major drawback of (pure) Monte Carlo methods is the variance of the results (optimal value and solutions of the approximate programs) in small-sample conditions.

Let us describe the Sample Average Approximation method (SAA) (Shapiro, 2003b), which uses Monte Carlo for generating the scenario tree. One starts by building the branching structure of the tree. Note that the method does not specify how to carry out that step. Practitioners often use the same branching factor for each node relative to a given decision stage. They also often concentrate the branchings at early stages: the branching factor is high at the root node and then decreases with the index of the decision stage. The next step of the method consists in sampling the node values according to the distributions conditioned on the values of the ancestor nodes. The procedure, referred to as *conditional sampling*, is implemented by sampling the realizations of random variables at stage t before sampling those of stage $t+1$. Distinct realizations are assigned to distinct nodes, which are given a conditional probability equal to the inverse of the branching factor. The last step consists in solving the program on the so-obtained scenario tree and thus, although part of the description of the SAA method, does not concern the generation of the tree itself.

Consider scenario trees obtained by conditional sampling. For simplicity assume a uniform branching factor n_t at each stage t, so that the number of scenarios is $n = \prod_{t=1}^{T} n_t$. Shapiro (2006) shows under some technical assumptions that if we want to guarantee, with a probability at least $1-\alpha$, that implementing the first-stage decision $\hat{u}_1$ optimized on a scenario tree of size n while implementing subsequently optimal recourse decisions conditionally to the first-stage decision will yield an objective value ϵ-close to the exact optimal value, then the size n of the tree we use for that purpose has to grow exponentially with the number of stages. The result goes against the intuition that by asking for ϵ-optimality with probability $1-\alpha$ only, one could get moderate sample complexity requirements. Now, as the exponential growth of the number of scenarios is not sustainable, one can only hope solving multistage models in small-sample conditions, and obtain solutions that at least with the SAA method may vary from tree to tree and be of uncertain value for the real problem. Perhaps surprisingly, it is not possible to obtain valid statistical bounds for that uncertain value by imposing as first-stage decision the tested first-stage decision and reoptimizing recourse decisions on several new random trees (Shapiro, 2003a).

Deterministic Scenario Tree Optimization Methods. There exists various deterministic techniques for selecting jointly the scenarios of the tree. Note that there is a part of numerical experimentation in the development of scenario tree methods, and a risk of overestimating the domain of validity of the proposed methods, since research efforts are oriented by experiments on particular problems.

Moment-matching methods (Høyland, Kaut, & Wallace, 2003) attempt to produce discrete distributions with some statistical moments matching those of a target distribution. Moment matching may be done at the expense of other statistics, such as the number and the location of the modes, that might also be important. Hochreiter and Pflug (2007) gives an example illustrating that risk.

The theoretical analysis underlying probability-metrics methods, that we have described in the context of two-stage stochastic programming, was initially believed to be easily extended to the multistage case (Heitsch & Römisch, 2003); but then it turned out that more elaborated measures of probability distances, integrating the intertemporal aspect of observations, were needed (Heitsch & Römisch, 2009). These elaborated metrics are more difficult to compute and to minimize, so that well-justified discretizations of multistage programs are more difficult to obtain.

We can also mention methods that come with approximation guarantees, such as bounds on the suboptimality of the approximation (Kuhn, 2005). However, they are applicable under very specific assumptions concerning the problem class and the type of randomness. Quasi Monte Carlo techniques are perhaps among the more generally applicable methods (Pennanen, 2009).

Most deterministic methods end up with the formulation of difficult optimization problems, such as nonconvex or NP-hard problems (Høyland et al., 2003; Hochreiter & Pflug, 2007), with computationally demanding tasks (such as multidimensional integrations), especially for high-dimensional random processes.

The field is still in a state where the scope of existing methods is not well defined, and where the algorithmic description of the methods is incomplete, especially concerning the branching structure of the trees. That the domains of applicability are not known or overestimated makes it delicate to select a sophisticated deterministic technique for building a scenario tree on a new problem.

4.3 The Need for Testing Scenario-Tree Approximations

Theoretical analyses of scenario tree generation algorithms, often based on worst-case reasonings or large deviation theory, provide guarantees on the quality of approximate solutions that are usually too loose in practice or equivalently call for intractable scenario tree sizes. Hence they do not really solve the basic question of how to build a priori small scenario trees in a generic, scalable, and computationally efficient way, potentially jeopardizing the practical relevance of the multistage extension of stochastic programming for sequential decision making under uncertainty. Now if we are ready to renounce to worst-case guarantees embedded in the scenario tree generation method, new tools are needed for computing, a posteriori, guarantees on the value of a given numerical approximation scheme.

If we want to assess on an independent test set of scenarios the performance of decisions optimized on a scenario tree, a difficulty arises: first-stage decisions can be tested but subsequent recourse decisions are only defined for the scenarios covered by the scenario tree. Therefore, it is necessary to extend the approach so as to allow one to test solutions on new scenarios, at a computational cost low enough to allow the validation on large numbers of test scenarios.

We have to stress that this extension is not really necessary for two-stage stochastic programming. First, approximations of two-stage models yield constant first-stage decisions, that are implementable on any scenario, while recourse decisions on new scenarios can then often be found analytically, or by running a myopic one-stage optimization procedure for each new scenario, or by implementing a known recourse procedure that the initial two-stage model was only approximating for optimizing the first-stage decisions — a strategy found efficient in capacity planning (Sen, Doverspike, & Cosares, 1994). Thus, testing is generally straightforward for two-stage models. Second, finite-dimensional approximations of two-stage stochastic programming models do not use scenario trees. They only use a finite set of outcomes. Theoretical results show that in the two-stage situation, statistical confidence bounds on the quality of an approximate solution can be computed (Mak, Morton, & Wood, 1999). These results break down in the multistage case, giving its true interest to guarantees based on testing (Shapiro, 2003a).

4.4 The Inference of Shrinking-Horizon Decision Policies

Several authors have proposed to use a generic scheme similar to Model Predictive Control to assess the performances associated to a particular algorithm $\mathcal{A}$ for building the scenario tree (Kouwenberg, 2001; Chiralaksanakul, 2003). The scheme can be sketched as follows.

1. Generate a scenario tree using algorithm $\mathcal{A}$, solve the resulting program, extract from its solution the first-stage decision u_1;
2. Generate a test sample of m mutually independent scenarios by sampling realizations of the random process.
3. For each scenario of the test sample, obtain sequentially the recourse decisions $u_2, \dots, u_T$, where each decision u_t is treated as a first-stage decision computed by taking as an initial condition the past decisions $u_1, \dots, u_{t-1}$ and the history $\xi_1, \dots, \xi_{t-1}$ of the test scenario, by conditioning the joint distribution of $\xi_t, \dots, \xi_T$ on the history, by using the algorithm $\mathcal{A}$ to build a new scenario tree that approximates the random process $\xi_t, \dots, \xi_T$, by solving the program formulated on this tree over the optimization variables relative to the decisions $u_t, \dots, u_T$, and by discarding all but the decision $u_t = \pi_t(\xi_1, \dots, \xi_{t-1})$.
4. Estimate the overall performance of the scheme on the test sample by forming the empirical average $V_{\text{TS}}(\mathcal{A}) = m^{-1} \sum_{j=1}^{m} f(\xi^j, u^j)$, where the sum runs over the indices relative to the scenarios in the test sample TS and their associated decision sequences $u^j = (u_1^j, \dots, u_T^j)$.

The statistical estimator $V_{\text{TS}}(\mathcal{A})$ provides an unbiased estimation of the value of the scenario tree building algorithm $\mathcal{A}$ in the context of the other approximations involved in the numerical computations of the sequences of decisions, such as simplifications of the objective function or early stopping at low-accuracy solutions.

The estimator may have a high variance, but we can expect a high positive correlation between estimators $V_{\mathrm{TS}}(\mathcal{A})$, $V_{\mathrm{TS}}(\mathcal{A}')$ relative to distinct algorithm variants $\mathcal{A}, \mathcal{A}'$ on the same test sample TS, allowing a reliable comparison of the relative performance of the two variants on the problem instance at hand.

The validation is generic in the sense that it can be applied to any algorithm $\mathcal{A}$, but also in the sense that it addresses the general scenario tree building problem in the larger context of the decision making scheme actually implemented in practice.

4.5 Synthesis

In the preceding subsections we have formulated and analyzed the problem of inferring a good scenario tree approximation for a given multistage stochastic programming problem. We have seen that the state of the art does not currently provide strong enough methods with broad enough practical coverage and good enough theoretical guarantees in terms of the quality of the approximate solutions derived in this way.

Researchers in the field were thus led to suggest the use of the shrinking-horizon recursive procedure for exploiting the scenario-tree based approach in practice. However, evaluating the resulting performance estimator on an independent sample of scenarios is extremely demanding, as it requires, for each test scenario and at each stage of recourse decisions, the automatic construction of a new scenario tree and the optimization of the resulting program on the tree. Doing this is still beyond the possibility of available computational approaches when considering the solution of large-scale problems.

For these reasons, there is currently no scalable off-the-shelf method for generating and testing scenario-tree based approximations of multistage stochastic programs, and the framework of stochastic programming based on scenarios trees has in this way, in spite of its theoretical appeal, lost its practical attractiveness during the last years in many environments dealing with large-scale systems (Powell & Topaloglu, 2003; Van Hentenryck & Bent, 2006).

5 MACHINE LEARNING BASED APPROACH

In this section we propose an approach for solving stochastic programming problems based on the idea of generating in a lazy fashion a large number of random tractable scenario-tree based approximations. The approach is lazy in the sense that instead of recommending a careful analysis of the structure of the problem at hand, and instead of devoting all computational resources to the construction of a single scenario tree, we recommend a multiplication of solution attempts through the generation of several approximations. The method works by extracting, from the solutions of these approximations, datasets that combine realizations of the random process and decision sequences, and by processing these datasets by a supervised learning method, so as to infer policies that can be later on tested efficiently on a large sample of new independent scenarios. These policies can be jointly exploited to infer multistage decision strategies that achieve good performances in a very generic way.

In this section, we describe our solution approach in general. Practical implementation details are easier to discuss on a concrete problem; this will be done in the next section.

5.1 Motivation

Our approach is motivated by two complementary and intimately related considerations induced by our analysis of the state of the art in approximation methods for stochastic programming, and their confrontation to the problems addressed in the field of machine learning in the last years.

The first motivation is derived from the need for intensive testing of decision-making policies for multistage programs. This need is primarily a consequence of the lack of tight theoretical results that would provide broadly usable, a priori guarantees on scenario-tree based methods. A posteriori testing of decisions by the shrinking-horizon approach is not a viable option, given its internal use of additional scenario trees, and its overall computational complexity. With respect to this motivation, machine learning offers a multitude of ways of extracting policies that are easy to test in an automatic way on a large number of independent samples. Keeping the sample set large is an unavoidable requirement for the statistical significance of performance estimators evaluated on high-dimensional random vectors and/or long scenarios, and thus ultimately for the practical use of the framework.

The second motivation has to do with the intrinsic nature of the finite scenario-tree approximation for multistage stochastic programming. The variance in the quality of the optimal decisions that may be inferred from finite approximations suggests that such problems are essentially *ill-posed* in the same sense as inverse problems addressed in machine learning are also ill-posed: small perturbations in the values of a finite data set used for the empirical estimation of an expectation leads to large variations (instability) of the predictor optimal with respect to the empirical expectation. Therefore, regularization techniques and principles from statistical learning theory (Vapnik, 1998), such as the structural risk minimization principle, may help to extract solutions from small scenario-tree approximations in a sound way from the theoretical point of view, and in an efficient way from the practical point of view.

To summarize the main ideas of the following subsections, we propose an approach that (i) allows to test small scenario trees quickly and reliably, (ii) is likely to offer better ways of exploiting individual scenario-tree approximations, and (iii) in the end allows to revisit the initial question of generating, solving, ranking and exploiting tractable scenario trees for solving complex multistage decision-making problems.

5.2 Inference and Evaluation of a Policy from a Scenario Tree

Cheap estimators of the quality of a scenario tree can be constructed by resorting to supervised learning techniques (Defourny et al., 2009). The basic principle consists in inferring a suboptimal policy by first learning a sequence of decision predictors $(\hat{\pi}_1, \dots \hat{\pi}_T)$ from a data set of examples of information state/decision pairs. The information states are extracted from the nodes of the scenario tree; they correspond to the partial scenario histories in the tree, but they could also be represented differently, for instance by features, or by a state in the sense of dynamic programming. The decisions are also extracted from the nodes of the tree; they are the optimal decisions computed on the tree. The predictions are then corrected in an ad-hoc fashion using a cheap repair procedure, denoted for simplicity by M_t, so as to obtain feasible decisions $u_t = M_t(\hat{\pi}_t(\xi_1, \dots, \xi_{t-1})) \in \mathcal{U}_t$

evaluated and implemented online on new scenarios. Repair procedures are also suggested in Küchler and Vigerske (2010) as a means of restoring the feasibility of decisions extracted from a tree and directly applied on test scenarios.

Formally, the decision predictor $\hat{\pi}_t$ is defined as a map from inputs $X_t = (\xi_1, \dots, \xi_{t-1})$ to outputs $Y_t = u_t$ learned from a dataset $D_t = \{(X_t^k, Y_t^k)\}_{k=1}^n$ of input-output pairs extracted from the scenario tree and its associated optimized decisions. The nature of the repair procedure varies with the feasibility constraints that should be enforced. It is necessary when standard supervised learning algorithms are unable to meet the constraints exactly. The decisions and observations on which $\mathcal{U}_t$ depend are passed in arguments of the repair procedure. An example of repair procedure is the projection of a predicted decision on the corresponding feasibility set. It is also possible to resort to simple problem-dependent heuristics for restoring feasibility.

This leads to the following inference and validation scheme:

1. Generate a scenario tree using an algorithm $\mathcal{A}$, solve the resulting program, extract from its solution the first-stage decision u_1, and the datasets D_t of partial-scenario/stage-t-decision pairs.
2. Learn the decision predictors $\hat{\pi}_t$ from the dataset D_t, for $t = 2, \dots, T$.
3. Generate a test sample of m mutually independent scenarios by sampling realizations ξ^j of the random process ξ.
4. For each scenario ξ^j of the test sample, set $u_1^j = u_1$ and obtain sequentially the recourse decisions $u_2^j, \dots, u_T^j$, where each decision u_t^j is obtained by first evaluating $\hat{\pi}_t(\xi_1^j, \dots, \xi_{t-1}^j)$ and then restoring feasibility by the repair procedure M_t.
5. Estimate the overall performance of the scheme on the test sample by forming the empirical average $V_{\mathrm{TS}}(\mathcal{A}) = m^{-1} \sum_{j=1}^m f(\xi^j, u^j)$, where the sum runs over the indices relative to the scenarios in the test sample and their associated decision sequences $u^j = (u_1^j, \dots, u_T^j)$.

The estimator $V_{\mathrm{TS}}(\mathcal{A})$ computed in this way reflects the quality of the scenario tree, the learned policy and the repair procedure. We expect however that scenario tree variants can be ranked reliably, despite the variance of the estimator due to the choice of the test sample, and despite a new source of bias due to the simplified representation of the decision policy by the supervised learning strategy. The value $V_{\mathrm{TS}}(\mathcal{A})$ is obtained by simulating an explicit policy that generates feasible decisions, and thus always provides a pessimistic bound (upper bound for minimization, lower bound for maximization) on the performance of the best policy that could be inferred from the current scenario tree. Hence a reliable bound on the achievable performance for the sequential decision making strategy is provided, up to the standard error of the test-sample estimator. (The standard error can be reduced by increasing m, under the assumption that the initial problem has a finite optimal value.) The pessimistic bound can be made tighter by testing various policies obtained from the same scenario tree but with different learning algorithms and/or repair procedures. The best combination of algorithms and learning parameters is then retained. In principle, the value of the best policy should be evaluated again on a new independent test sample.

Note that a learned policy is not necessarily always worse than a shrinking-horizon policy using the same first-stage decision u_1, as the regularization that occurs during the supervised learning step could actually improve the quality of the recourse decisions $u_2, \dots, u_T$.

Note also that the input space of the learned policy is a simple matter of convenience. As long as the policy remains non-anticipative, the input space can be reparameterized, typically by letting appear explicitly past decisions, state variables, and additional features derived from the information state and that might facilitate the generalization of the decisions in the data sets, or later, the online evaluation of the learned predictors.

5.3 Simple Monte Carlo Based Search of Scenario Trees

We are now ready to sketch a workable and generic scheme for obtaining approximate solutions to multistage stochastic programs with performance guarantees. The scheme builds on the procedure of the previous section, which allows to extract a policy from any scenario tree and from optimal solutions for the approximate program associated with the tree, and allows to estimate the value of the policy by Monte Carlo simulation with a sufficient accuracy. In its most elementary version, the scheme consists in generating a possibly large set of randomized scenario-tree approximations for a given problem $\mathcal{S}$, ranking them according to the estimated value of their corresponding policy, and identifying in this way automatically the best scenario tree among the considered sample of trees.

More specifically, we propose the following algorithm.

1. Generate a large test sample TS of mutually independent scenarios for the problem under consideration.
2. Generate the branching structure for a scenario tree randomly. The procedure may use any valuable insight on good branching structures (in general or specific to the problem).
3. Use an existing scenario tree generation method to instantiate the values and probabilities of the nodes of the tree, for example, the Monte Carlo based method SAA already described. That the generation of the values defining the scenarios could be integrated to the construction of the branching structure is not really relevant: the two phases are separated here to stress that they are carried out independently. The construction of the tree is independent of the test sample TS.
4. Solve the approximate program derived from the so-obtained scenario tree.
5. Extract from the solution the datasets of information state/decision pairs that are necessary to learn a suboptimal policy for the recourse decisions, and then associate to the scenario tree the score V_{TS} corresponding to the average performance of that policy on the test sample.
6. Return to step 2 until a stopping criterion is met. Output the first-stage decision u_1 from the scenario tree with the best score, and the guarantee that u_1 yields at least $V^*_{\mathrm{TS'}}$ up to some standard error, with the corresponding learned policy as a certificate. Here $V^*_{\mathrm{TS'}}$ is ideally the estimation of the value of the best policy on a new, very large test sample TS' of mutually independent scenarios.

As this procedure performs a kind of model selection on the basis of scores measured on the test sample TS, an unbiased estimation of the performance of the selected first-stage decision u_1 and associated recourse policy calls for a new independent test sample TS'. The initial test sample TS should then be called a *validation sample*, according to the standard machine learning terminology. In our numerical experiments, we merely used a very large validation sample TS in order to evaluate various policies and estimate their respective performances in a reliable way.

In applications where the only information about the random process is a finite set of realizations, the method could be extended as follows: one would split the set of realizations into a test set serving as the test sample TS, and a learning set LS from which a generative model $\mathcal{G}_{\mathrm{LS}}$ for the random process would be inferred, that is, a best possible estimate for $\mathbb{P}$. The scenario trees would then be built by querying new samples from the generative model.

5.4 Discussion

The generic procedure presented in this section is based on various open ingredients that may be largely exploited for the design of a wide class of algorithms in a very flexible way. Namely, the “meta-parameters” are (i) the scenario tree sampling scheme, (ii) the (possibly regularized) optimization technique used to extract from a scenario tree a dataset, (iii) the precise supervised learning algorithm used to obtain the decision strategies from the datasets, (iv) the repair procedure used to restore the feasibility of the decisions on new scenarios.

The main ideas of the proposed scheme are evaluated in the case study section on a family of problems proposed by other authors. We illustrate how one may adjust the scenario tree generation algorithm and the policy learning algorithm to one’s needs, and by doing so we also illustrate the flexibility of the proposed approach and the potential of the combination of scenario-tree based decision making with supervised learning. In particular, the efficiency of supervised learning strategies makes it possible to rank large numbers of policies inferred from large numbers of randomly generated scenario trees.

Although we do not illustrate this in the present work, we would like also to stress that the scenario tree sampling scheme may be coupled in various other ways with the inference of policies by machine learning. For example, one could seek to use sequential Monte Carlo techniques inspired from the importance sampling literature, in order to progressively guide the scenario tree sampling and machine learning methods towards regions of high interest, given the quality of the policies inferred from scenarios trees at previous iterations. Also, instead of using each dataset obtained from each scenario tree to extract a single policy, one could extract multiple policies from a single dataset, or use several datasets and learning algorithms to extract a single policy, in the spirit of the wide range of model combination and perturbation schemes from the machine learning literature (Dietterich, 2000).

6 CASE STUDY

We will show the interest of the approximate solution techniques presented in the chapter by applying them to a family of multistage stochastic programs. Implementation choices difficult to discuss in general terms, such as choices concerning the supervised learning of a policy for the recourse decisions, and the choices for the random generation of the trees, will be illustrated on a concrete case.

The section starts by the formulation of a multistage stochastic program that various researchers have presented as difficult for scenario tree methods (Hilli & Pennanen, 2008; Koivu & Pennanen, 2010; Küchler & Vigerske, 2010). Several instances of the problem will be addressed, including instances on horizons considered as almost unmanageable by scenario tree methods.

6.1 Problem Description

We consider a multistage problem adapted from Hilli and Pennanen (2008), interpreted in that paper as the valuation of an electricity swing option. In this chapter, we interpret the problem rather as the search for risk-aware strategies for distributing the sales of a commodity over T stages in a flexible way adapted to market prices. A risk-aware objective is very interesting for our purposes, but it is difficult to justify it in a context of option valuation. The formulation of the problem is as follows:

$$\begin{array}{ll} \text{minimize} & \rho^{-1} \log \mathbb{E}\{\exp\{-\rho \sum_{t=1}^{T} \xi_{t-1} \cdot \pi_t(\xi)\}\} \\ \text{subject to} & 0 \leq \pi_t(\xi) \leq 1 \text{ and } \sum_{t=1}^{T} \pi_t(\xi) \leq Q \ , \\ & \pi \ \text{ non-anticipative.} \end{array}$$

The objective uses the exponential utility function, with risk aversion coefficient ρ. Such objectives are discussed at the end of the chapter.

In our formulation of the problem, there is no constant first-stage decision to optimize. We begin directly by the observation of ξ_0, followed by a recourse decision $u_1 = \pi_1(\xi_0)$. Observations and decisions are intertwined so that in general $u_t = \pi_t(\xi_0, \ldots, \xi_{t-1})$. The random variable ξ_{t-1} is the unitary profit ($\xi_{t-1} > 0$) or loss ($\xi_{t-1} < 0$) that can result from the sale of the commodity at time t. Potential profits and losses fluctuate in time, depending on market conditions (we later select a random process model for market prices to complete the problem specification). The commodity is sold in quantity $u_t = \pi_t(\xi_0, \ldots, \xi_{t-1})$ at time t, meaning that the quantity u_t can depend on past and current prices. The decision is made under the knowledge of the potential profit or loss at time t, given by $\xi_{t-1} \cdot u_t$, but under uncertainty of future prices. This is by the way why scenario tree techniques must be used with great care on this problem when the planning horizon is long: as soon as the scenarios cease to have branchings, there is no more residual uncertainty on future prices, and the optimization process wrongly identifies opportunities anticipatively. Those spurious future opportunities may significantly degrade the quality of previous decisions.

We seek strategies where the sales per stage are bounded (constraint $0 \leq \pi_t(\xi) \leq 1$). The constraint can model a bottleneck in the production process. Notice also that bounded sales are consistent with the model assumption of an exogenous random process: very large sales are more likely to influence the market prices on long planning horizons. The scalar Q bounds the total sales (we assume $Q \geq 1$). It represents the initial stock of commodity, the sale of which must be distributed optimally over the horizon T.

When the risk aversion coefficient ρ tends to 0, the problem reduces to the search of a risk-neutral strategy. This case has been studied by Küchler and

Vigerske (2010). It admits a linear programming formulation:

$$\begin{array}{ll} \text{minimize} & -\mathbb{E}\{\sum_{t=1}^{T} \xi_{t-1} \cdot \pi_t(\xi)\} \\ \text{subject to} & 0 \leq \pi_t(\xi) \leq 1 \text{ and } \sum_{t=1}^{T} \pi_t(\xi) \leq Q \ , \\ & \pi \ \text{ non-anticipative,} \end{array}$$

and an exact analytical solution (which thus serves as a reference)

$$\pi_t^{\mathrm{ref}}(\xi) = \begin{cases} 0 & \text{if } t \leq T - Q \text{ or } \xi_{t-1} \leq 0 \ , \\ 1 & \text{if } t > T - Q \text{ and } \xi_{t-1} > 0 \ . \end{cases}$$

In a first experiment, we will take the numerical parameters and the process ξ selected in Hilli and Pennanen (2008) (to ease the comparisons): $\rho = 1$, $T = 4$, $Q = 2$; $\xi_t = (\exp\{b_t\} - K)$ where $K = 1$ is the fixed cost (or the strike price, when the problem is interpreted as the valuation of an option) and b_t is a random walk: $b_0 = \sigma \ \epsilon_0$, $b_t = b_{t-1} + \sigma \ \epsilon_t$, with $\sigma = \sqrt{0.2}$ and ϵ_t following a standard normal distribution $\mathcal{N}(0, 1)$.

In a second experiment over various values of the parameters (ρ, Q, T) with T up to 52, we will take for ξ the process selected in Küchler and Vigerske (2010) (because otherwise on long horizons the price levels of the first process blow out in an unrealistic way, making the problem rather trivial): $\xi_t = (\xi'_t - K)$ with $\xi'_t = \xi'_{t-1} \exp\{\sigma\epsilon_t - \sigma^2/2\}$ where $\sigma = 0.07$, $K = 1$, and ϵ_t following a standard normal distribution. Equivalently $\xi_t = (\exp\{b_t - (t+1) \ \sigma^2/2\} - K)$ with b_t a random walk such that $b_0 = \sigma \ \epsilon_0$ and $b_t = b_{t-1} + \sigma \ \epsilon_t$.

6.2 Algorithm for Generating Small Scenario Trees

At the heart of tree selection procedure relies our ability to generate scenario trees reduced to a very small number of scenarios, with interesting branching structures. As the trees are small, they can be solved quickly and then scored using the supervised learning policy inference procedure. Fast testing procedures make it possible to rank large numbers of random trees.

The generation of random branching structures has not been explored in the classical stochastic programming literature; we thus have to propose a first algorithm in this section. The algorithm is developed with our needs in view, with the feedback provided by the final numerical results of the tests, until results on the whole set of considered numerical instances suggest that the algorithm suffices for the application at hand. We believe that the main ideas behind this algorithm will be reused in subsequent work for addressing the representation of stochastic processes of higher dimensions. Therefore, in the following explanations we put more emphasis on the methodology we followed than on the final resulting algorithm.

Method of Investigation. The branching structure is generated by simulating the evolution of a branching process. We will soon describe the branching process that we have used, but observe first that the probability space behind the random generation of the tree structure is not at all related to the probability space of the random process that the tree approximates. It is the values and probabilities of the nodes that are later chosen in accordance to the target probability distribution, either deterministically or randomly, using any new or existing method.

For selecting the node values, we have tested different deterministic quantizations of the one-dimensional continuous distributions of random variables ξ_t, and alternatively different quantizations of the gaussian innovations ϵ_t that serve to define $\xi_t = \xi_t(\epsilon_t)$, as described by the relations given in the previous section. Namely, we have tested the minimization of the quadratic distortion (Pages & Printems, 2003) and the minimization of the Wasserstein distance (Hochreiter & Pflug, 2007). On the considered problems we did not notice significant differences in performance attributable to a particular deterministic variant.

What happened was that with deterministic methods, performances began to degrade as the planning horizon was increased, perhaps because trying to preserve statistical properties of the marginal distributions ξ_t distorts other statistics of the joint distribution of $(\xi_0, \dots, \xi_{T-1})$, especially in higher dimensions. Therefore, for treating instances on longer planning horizons, we switched to a crude Monte Carlo sampling for generating node values.

By examining trees with the best scores in the context of the present family of problems, we observed that several statistics of the random process represented by those trees could be very far from their theoretical values, including first moments. This might suggest that it is very difficult to predict without any information on the optimal solutions which properties should be preserved in small scenario trees, and thus which objective should be optimized when attempting to build a small scenario tree. If we had discovered a correlation between some features of the trees and the scores, we could have filtered out bad trees without actually solving the programs associated to these trees, simply by computing the identified features.

Description of the Branching Processes. We now describe the branching process used in the experiments made with deterministic node values. Let $r \in [0, 1]$ denote a fixed probability of creating a branching. We start by creating the root node of the tree (depth 0), to which we assign the conditional probability 1. With probability r, we create 2 successor nodes to which we assign the values ± 0.6745 and the conditional probabilities 0.5 (the values given here minimize the Wasserstein distance between a two mass point distribution and the standard normal distribution for ϵ_t). With probability $(1 - r)$ we create instead a single successor node to which we assign the value 0 and the conditional probability 1; this node is a degenerate approximation of the distribution of ϵ_t. Then we take each node of depth 1 as a new root and repeat the process of creating 1 or 2 successor nodes to these new roots randomly. The process is further repeated on the nodes of depth $2, \dots, T - 1$, yielding a tree of depth T for representing the original process $\epsilon_0, \dots, \epsilon_{T-1}$. The scenario tree for ξ is derived from the scenario tree for ϵ.

For problems on larger horizons, it is difficult to keep the size of the tree under control with a single fixed branching parameter r — the number of scenarios would have a large variance. Therefore, we used a slightly more complicated branching process, by letting the branching probability r depend on the number of scenarios currently developed. Specifically, let N be a target number of scenarios and T a target depth for the scenario tree with the realizations of ξ_t relative to depth $t+1$. Let n_t be the number of parent nodes at depth t; this is a random variable except at the root for which $n_0 = 1$. During the construction of the tree, parent nodes at depth $t < T$ are developed and split in $\nu = 2$ children nodes with a probability $r_t = n_t^{-1}(\nu-1)^{-1}(N-1)/T$. Parent nodes have a single child node with a probability $1-r_t$. If $r_t > 1$, we set $r_t = 1$ and all nodes are split in $\nu = 2$ children nodes. Thus in general $r_t = \min\{1,\ n_t^{-1}(\nu-1)^{-1}(N-1)/T\}$.

6.3 Algorithm for Learning Policies

Solving a program on a scenario tree yields a dataset of scenario/decision sequence pairs (ξ, u). To infer a decision policy that generalizes the decisions of the tree to test scenarios, we have to learn mappings from $(\xi_0, \ldots, \xi_{t-1})$ to u_t and ensure the compliance of the decisions with the constraints. To some extent the procedure is thus problem-specific. Here again we insist on the methodology.

Dimensionality Reduction. We try to reduce the number of features of the input space. In particular, we can try to get back to a state-action space representation of the policy (and postprocess datasets accordingly to recover the states). Note that in general needed states are those that would be used by an hypothetical reformulation of the optimization problem using dynamic programming. Here the objective is based on the exponential utility function. By the property that

$$\begin{aligned}&\mathbb{E}\{\exp\{-\textstyle\sum_{t'=1}^{T} \xi_{t'-1} \cdot u_{t'}\} \mid \xi_0, \ldots, \xi_{t-1}\} \\ &= \exp\{-\textstyle\sum_{t'=1}^{t-1} \xi_{t'-1} \cdot u_{t'}\}\ \mathbb{E}\{\exp\{-\textstyle\sum_{t'=t}^{T} \xi_{t'-1} \cdot u_{t'}\} \mid \xi_0, \ldots, \xi_{t-1}\}\ ,\end{aligned}$$

we can see that decisions at $t' = 1, \ldots, t-1$ scale by a same factor the contribution to the return brought by the decisions at $t' = t, \ldots, T$. Therefore, if the feasibility set at time t can be expressed from state variables, the decisions at $t' = t, \ldots, T$ can be optimized independently of the decisions at $t' = 1, \ldots, t-1$. This suggests to express u_t as a function of the state ξ_{t-1} of the process ξ, and of an additional state variable ζ_t defined by $\zeta_0 := Q$, $\zeta_t := Q - \sum_{t'=1}^{t-1} u_{t'}$, that allows to reformulate the constraint $\sum_{t'=1}^{T} u_{t'} \leq Q$ as $\sum_{t'=t}^{T} u_{t'} \leq \zeta_t$.

Feasibility Guarantees Sought Before Repair Procedures. We try to map the output space in such a way that the predictions learned under the new geometry and then transformed back using the inverse mapping comply with the feasibility constraints. Here, we scale the output u_t so as to have to learn the fraction $y_t = y_t(\xi_{t-1}, \zeta_t)$ of the maximal allowed output $\min(1, \zeta_t)$, which summarizes the two constraints of the problem. Since $\zeta_0 = Q$, we distinguish the cases $u_1 = y_1(\xi_0) \cdot 1$ and $u_t = y_t(\xi_{t-1}, \zeta_t) \cdot \min(1, \zeta_t)$. It will be easy to ensure that fractions y_t are valued in $[0, 1]$ (thus we do not need to define an a posteriori repair procedure).

Input Normalization. It is convenient for the sequel to normalize the inputs. From the definition of ξ_{t-1} we can recover the state of the random walk b_{t-1}, and use as first input $x_{t1} := (\sigma^2 t)^{-1/2} b_{t-1}$, which follows a standard normal distribution. Thus for the first version of the process ξ, instead of ξ_{t-1} we use $x_{t1} = \sigma^{-1} t^{-1/2} \log(\xi_{t-1} + K)$, and for the second version of the process ξ, instead of ξ_{t-1} we use $x_{t1} = \sigma^{-1}\, t^{-1/2} \log(\xi_{t-1} + K) + \sigma t^{1/2}/2$. Instead of the second input ζ_t (for $t > 1$) we use $x_{t2} := \zeta_t / Q$, which is valued in $[0, 1]$. Therefore, we rewrite the fraction $y_t(\xi_{t-1}, \zeta_t)$ as $g_t(x_{t1}, x_{t2})$.

Hypothesis Space. We have to choose the hypothesis space for the learned fractions g_t defined previously. Here we find it convenient to choose the class of feed-forward neural networks with one hidden layer of L neurons:

$$g_t(x_{t1}, x_{t2}) = \text{logsig}\left(\gamma_t + \textstyle\sum_{j=1}^{L} w_{tj} \cdot \text{tansig}\left(\beta_{tj} + \sum_{k=1}^{2} v_{tjk}\ x_{tk}\right)\right) ,$$

with weights v_{tjk} and w_{tj}, biases β_{tj} and γ_t, and activation functions

$$\begin{aligned} \text{tansig}(x) &= 2 \cdot (1 + e^{-2\,x})^{-1} - 1 && \text{valued in } [-1, +1]\ , \\ \text{logsig}(x) &= (1 + e^{-x})^{-1} && \text{valued in } [0, 1]\ , \end{aligned}$$

a usual choice for imposing the output ranges $[-1, +1]$ and $[0, 1]$ respectively.

Since the training sets are extremely small, we take $L = 2$ for g_1 (which has only one input x_{11}) and $L = 3$ for g_t $(t > 1)$.

We recall that artificial neural networks have been found to be well-adapted to nonlinear regression. Standard implementations of neural networks (construction and learning algorithms) are widely available, with a full documentation from theory to interactive demonstrations (Demuth & Beale, 1993). We report here the parameters chosen in our experiments for the sake of completeness; the method is largely off-the-shelf.

Details on the Implementation. The weights and biases are determined by training the neural networks. We used the Neural Network toolbox of Matlab with the default methods for training the networks by backpropagation — the Nguyen-Widrow method for initializing the weights and biases of the networks randomly, the mean square error loss function, and the Levenberg-Marquardt optimization algorithm. We used $[-3, 3]$ for the estimated range of x_{t1}, corresponding to 3 standard deviations, and $[0, 1]$ for the estimated range of x_{t2}.

Trained neural networks are dependent on the initial weights and biases before training, because the loss minimization problem is nonconvex. Therefore, we repeat the training 5 times from different random initializations. We obtain several candidate policies (to be ranked on the test sample). In our experiments on the problem with $T = 4$, we randomize the initial weights and biases of each network independently. In our experiments on problems with $T > 4$, we randomize the initial weights and biases of $g_1(x_{11})$ and $g_2(x_{21}, x_{22})$, but then we use the optimized weights and biases of g_{t-1} as the initial weights and bias for the training of g_t. Such a warm-start strategy accelerates the learning tasks. Our intuition was that for optimal control problems, the decision rules π_t would change rather slowly with t, at least for stages far from the terminal horizon.

We do not claim that using neural networks is the only or the best way of building models g_t that generalize well and are fast in exploitation mode. The choice of the Matlab implementation for the neural networks could also be criticized. It just turns out that these choices are satisfactory in terms of implementation efforts, reliability of the codes, solution quality, and overall running time.

6.4 Remark on Approximate Solutions

An option of the proposed testing framework that we have not discussed, as it is linked to technical aspects of numerical optimization, is that we can form the datasets of scenario/decisions pairs using inexact solutions to the optimization programs associated to the trees. Indeed, simulating a policy based on any dataset will still give a pessimistic bound on the optimal solution of the targeted problem. The tree selection procedure will implicitly take this new source of approximation into account. In fact, every approximation one can think of for solving the programs could be tested on the problem at hand and thus ultimately accepted or rejected, on the basis of the performance of the policy on the test sample, and the time taken by the solver to generate the decisions of the dataset.

6.5 Numerical Results

We now describe the numerical experiments we have carried out and comment on the results.

Experiment on the small-horizon problem instance. First, we consider the process ξ and parameters (ρ, Q, T) taken from Hilli and Pennanen (2008). We generate a sample of $m = 10^4$ scenarios drawn independently, on which each learned policy will be tested. We generate 200 random tree structures as described previously (using $r = 0.5$ and rejecting structures with less than 2 or more than 10 scenarios). Node values are set by the deterministic method, thus the variance in performance that we will observe among trees of similar complexity will come mainly from the branching structure. We form and solve the programs on the trees using cvx, and extract the datasets. We generate 5 policies per tree, by repeatedly training the neural networks from random initial weights and biases. Each policy is simulated on the test sample and the best of the 5 policies is retained for each tree.

The result of the experiment is shown on Figure 3. Each point is relative to a particular scenario tree. Points from left to right are relative to trees of increasing size. We report the value of $m^{-1}\sum_{j=1}^{m}\exp\{-\sum_{t=1}^{T}\xi_{t-1}^{j}\cdot\hat{\pi}_t(\xi^j)\}$ for each learned policy $\hat{\pi}$, in accordance with the objective minimized in Hilli and Pennanen (2008). Lower is better. Notice the large variance of the test sample scores among trees with the same number of scenarios but different branching structures.

The tree selection method requires a single lucky outlier to output a good valid upper bound on the targeted objective — quite an advantage with respect to approaches based on worst-case reasonings for building a single scenario tree. With a particular tree of 6 scenarios (best result: 0.59) we already reach the guarantee that the optimal value of our targeted problem is less or equal to $\log(0.59) \simeq -0.5276$. On Figure 4, we have represented graphically some of the lucky small scenario trees associated to the best performances. Of course, tree structures that perform well here may not be optimal for other problem instances.

The full experiment takes 10 minutes to run on a pc with a single 1.55 GHz processor and 512 Mb RAM. By comparing our bounds to those reported in Hilli and Pennanen (2008) — where validation experiments taking up to 30 hours with

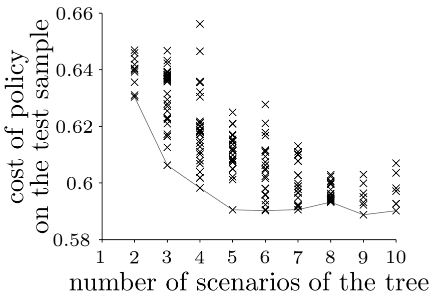

Figure 3. First experiment: scores on the test sample associated to the random scenario trees (lower is better). The linear segments join the best scores of policies inferred from trees of equivalent complexity.

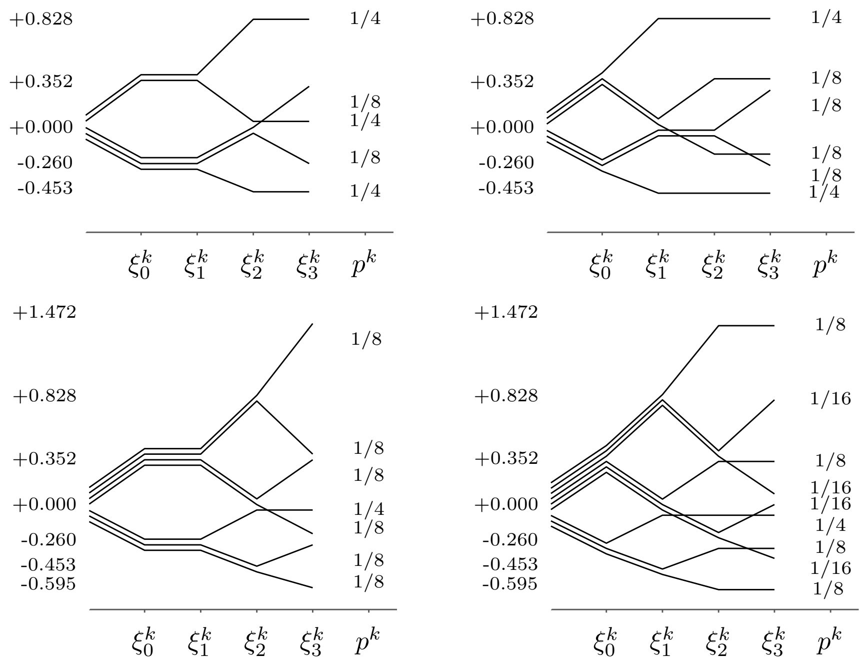

Figure 4. Small trees (5,6,7,9 scenarios) from which good datasets could be obtained. The scenarios $\xi^k = (\xi_0^k, \xi_1^k, \xi_2^k)$ are shifted vertically to distinguish them when they pass through common values, written on the left. Scenario probabilities p^k are indicated on the right.

a single 3.8Ghz processor, 8Gb RAM have been carried out — we deduce that we reached essentially the quality of the optimal solution.

Experiment on large-horizon problem instances. Second, we consider the process ξ taken from Küchler and Vigerske (2010) and a series of 15 sets of parameters for (ρ, Q, T). We repeat the following experiment on each (ρ, Q, T) with 3 different parameter values for controlling the size of the random trees:

Table 2. Second experiment: Best upper bounds for a family of problem instances.

Problem			Upper bounds[1]			
ρ	Q	T	Reference[2]	Value of the best policy[3], in function of N		
				$N = 1 \cdot T$	$N = 5 \cdot T$	$N = 25 \cdot T$
0	2	12	-0.1869	-0.1837	-0.1748	-0.1797
	2	52	-0.4047	-0.3418	-0.3176	-0.3938
	6	12	-0.5062	-0.5041	-0.4889	-0.4930
	6	52	-1.1890	-1.0747	-1.0332	-1.1764
	20	52	-3.6380	-3.5867	-3.5000	-3.4980
0.25	2	12	-0.1750	-0.1716	-0.1661	-0.1700
	2	52	-0.3351	-0.3210	-0.3092	-0.3288
	6	12	-0.4363	-0.4371	-0.4381	-0.4365
	6	52	-0.7521	-0.7797	-0.7787	-0.8030
	20	52	-1.4625	-1.8923	-1.9278	-1.9128
1	2	12	-0.1466	-0.1488	-0.1473	-0.1458
	2	52	-0.2233	-0.2469	-0.2222	-0.2403
	6	12	-0.3078	-0.3351	-0.3385	-0.3443
	6	52	-0.3676	-0.5338	-0.5291	-0.5354
	20	52	-0.5665	-0.9625	-0.9757	-0.9624

[1] Estimated on a test sample of $m = 10000$ scenarios. On a same row, lower is better.
[2] Defined by $\pi_t^{\mathrm{ref}}(\xi)$ and optimal for the risk-neutral case $\rho = 0$.
[3] On random trees of approximately N scenarios.

generate 25 random trees (we recall that this time the node values are also randomized), solve the resulting 25 programs, learn 5 policies per tree (depending on the random initialization of the neural networks), and report as the best score the lowest of the resulting 125 values computed on the test sample.

Table 2 reports values corresponding to the average performance

$$\rho^{-1} \log\{m^{-1} \sum_{j=1}^{m} \exp\{-\rho \sum_{t=1}^{T} \xi_{t-1}^{j} \cdot \hat{\pi}_t(\xi^j)\}\}$$

obtained for a series of problem instances, the numerical parameters of which are given in the first column of the table, for different policies selected among random trees of 3 different nominal sizes, so as to investigate the effect of the size of the tree on the performance of the learned policies. One column is dedicated to the performance of the analytical reference policy π^{ref} on the test sample.

In the case $\rho = 0$, the reference value provided by the analytical optimal policy suggests that the best policies are close to optimality. In the case $\rho = 0.25$, the reference policy is now suboptimal. It still slightly dominates the learned policies when $Q = 2$, but not anymore when $Q = 6$ or $Q = 20$. In the case $\rho = 1$, the reference policy is dominated by the learned policies, except perhaps when $Q = 2$ and the trees are large. That smaller trees are sometimes better than large trees may be explained by the observation that multiplying the number of scenarios by 25, as done in our experiments, does not fundamentally change the order of magnitude of size of the tree, given the required exponential growth of the number of scenarios with the number of stages.

This experiment shows that even if the scenario tree selection method requires generating and solving several trees, rather than one single tree, it can work very well. In fact, with a random tree generation process that can generate a medium size set of very small trees, there is a good likelihood in the problem that at least one of those trees will lead to excellent performances. Large sets of scenario trees could easily be processed simply by parallelizing the tree selection procedure. Overall, the approach seems promising in terms of the usage of computational resources.

7 TIME INCONSISTENCY AND BOUNDED RATIONALITY LIMITATIONS

This section discusses the notion of dynamically consistent decision process, which is relevant to sequential decision making with risk-sensitivity — by opposition to the optimization of the expectation of a total return over the planning horizon, which can be described as risk-indifferent, or risk-neutral.

7.1 Time-Consistent Decision Processes

We will say that an objective induces a dynamically consistent policy, or time-consistent policy, if the decisions selected by a policy optimal for that objective coincide with the decisions selected by a policy recomputed at any subsequent time step t and optimal for the same objective with decisions and observations prior to t set to their realized value (and decisions prior to t chosen according to the initial optimal policy).

Time-consistent policies are not necessarily time-invariant: we simply require that the optimal mappings π_t from information states i_t to decisions u_t at time t, evaluated from some initial information state at $t = 0$, do not change if we take some decisions following these mappings, and then decide to recompute them from the current information state. We recall that in the Markov Decision Process framework, the information state i_t is the current state x_t, and in the multistage stochastic programming framework, i_t is the current history $(\xi_1, \dots, \xi_{t-1})$ of the random process, with t indexing decision stages. We say that a decision process is time-consistent if it is generated by a time-consistent policy.

A close notion of time-consistency can also be defined by saying that the preferences of the decision maker among possible distributions for the total return over the planning horizon can never be affected by future information states that the agent recognizes, at some point in the decision process, as impossible to reach (Shapiro, 2009; Defourny, Ernst, & Wehenkel, 2008).

In the absence of time-consistency, the following situation may arise (the discussion is made in the multistage stochastic programming framework). At time $t = 1$, an agent determines that for each possible outcome of a random variable ξ_2 at time $t = 2$, the decision $u_2 = a$ at time $t = 2$ is optimal (with respect to the stated objective and constraints of the problem, given the distribution of $\xi_2, \xi_3, \dots$, and taking account of optimized recourse decisions $u_3, u_4, \dots$ over the planning horizon). Then at time $t = 2$, having observed the outcome of the random variable ξ_1 and conditioned the probability distributions of $\xi_2, \xi_3, \dots$ over this observation, and in particular, having ruled out all scenarios where ξ_1 differs from the observed outcome, the agent finds that for some possible realizations of ξ_2, $u_2 = a$ is not optimal.

The notion of time-consistency already appears in Samuelson (1937), who states: “as the individual moves along in time there is a sort of perspective phenomenon in that his view of the future in relation to his instantaneous time position remains invariant, rather than his evaluation of any particular year” (page 160). Several economists have rediscovered and refined the notion (Strotz, 1955; Kydland & Prescott, 1977), especially when trying to apply expected utility theory, valid for comparisons of return distributions viewed from a single initial information state, to sequential decision making settings, where the information state evolves.

In fact, if an objective function subject to constraints can be optimized by dynamic programming, in the sense that a recursive formulation of the optimization is possible using value functions (on an augmented state space if necessary, and irrespectively of complexity issues), then an optimal policy will satisfy the time-consistency property. This connection between Bellman’s principle (1957) and time-consistency is well-established (Epstein & Schneider, 2003; Artzner, Delbaen, Eber, Heath, & Ku, 2007). By definition and by recursion, a value function is not affected by states that have a zero probability to be reached in the future; when the value function is exploited, a decision u_t depends only on the current information state i_t. Objectives that can be optimized recursively include the expected sum of rewards, and the expected exponential utility of a sum of rewards (Howard & Matheson, 1972), with discount permitted, although the recursion gets more involved (Chung & Sobel, 1987). A typical example of objective that cannot be rewritten recursively in general is the variance of the total return over several decision steps. This holds true even if the state fully describes the distribution of total returns conditionally to the current state. Note, however, that a nice way of handling a mean-variance objective on the total return is to relate it to the expected exponential utility: if R denotes a random total return, $\Phi_\rho\{R\} = \mathbb{E}\{R\} - (\rho/2)\mathrm{var}\{R\} \simeq -\rho^{-1} \log \mathbb{E}\{\exp(-\rho R)\}$. The approximation holds for small $\rho > 0$. It is exact for all $\rho > 0$ if R follows a Gaussian distribution.

7.2 Limitations of Validations Based on Learned Policies

In our presentation of multistage stochastic programming, we did not discuss several extensions that can be used to incorporate risk awareness in the decision making process. In particular, a whole branch of stochastic programming is concerned with the incorporation of *chance constraints* in models (Prékopa, 1995), that is, constraints to be satisfied with a probability less than 1. Another line of research involves the incorporation of modern risk measures such as the *conditional value-at-risk* at level α (expectation of the returns relative to the worst α-quantile of the distribution of returns) (Rockafellar & Uryasev, 2000). An issue raised by many of these extensions, when applied to sequential decision making, is that they may induce time-inconsistent decision making processes (Boda & Filar, 2006).

The validation techniques based on supervised learning that we have proposed are not adapted to time-inconsistent processes. Indeed, these techniques rely on the assumption that the optimal solution of a multistage stochastic program is a sequence of optimal mappings π_t from reachable information states

$(\xi_1, \ldots, \xi_{t-1})$ to feasible decisions u_t, uniquely determined by some initial information state at which the optimization of the mappings takes place. We believe, however, that the inability to address the full range of possible multistage programming models should have minor practical consequences. On the one hand, we hardly see the point of formulating a sophisticated multistage model with optimal recourse decisions unrelated to those that would be implemented if the corresponding information states are actually reached. On the other hand, it is always possible to simulate any learned policy, whatever the multistage model generating the learning data might be, and score an empirical return distribution obtained with the simulated policy according to any risk measure important for the application. Computing a policy and sticking to it, even if preferences are changing over time, is a form of precommitment (Hammond, 1976).

Finally, let us observe that a shrinking-horizon policy can be time-inconsistent for two reasons: (i) the policy is based on an objective that cannot induce a time-consistent decision process; (ii) the policy is based on an objective that could be reformulated using value functions, but anyway the implicit evaluation of these value functions changes over time, due to numerical approximations local to the current information state. Similarly, if an agent uses a supervised-learning based policy to take decisions at some stage and is then allowed to reemploy the learning procedure at later stages, the overall decision sequence may appear as dynamically inconsistent. The source (ii) of inconsistency appears rather unavoidable in a context of bounded computational resources; more generally, it seems that bounded rationality (Simon, 1956) would necessarily entail dynamical inconsistency.

8 CONCLUSIONS

In this chapter, we have presented the principles of the multistage stochastic programming approach to sequential decision making under uncertainty, and discussed the inference and exploitation of decision policies for comparing various approximations of a multistage program in the absence of tight theoretical guarantees.

Sequential decision making problems under uncertainty form a rich class of optimization problems with many challenging aspects. Markov Decision Processes and multistage stochastic programming are two frameworks for addressing such problems. They have been originally studied by different communities, leading to a separate development of new approximation and solution techniques. In both fields, research is done so as to extend the scope of the framework to new problem classes: in stochastic programming, there is research on robust approaches (Delage & Ye, 2008), decision-dependent random processes (Goel & Grossmann, 2006), nonconvex problems (Dentcheva & Römisch, 2004); in Markov Decision Processes, many efforts are directed at scaling dynamic programming (or policy search) to problems with high-dimensional continuous state spaces and/or decision spaces (Ng & Jordan, 1999; Ghavamzadeh & Engel, 2007; Antos, Munos, & Szepesvári, 2008).

It is likely that a better integration of the ideas developed in the two fields will ultimately yield better solving strategies for large-scale problems having both continuous and discrete aspects. Both fields have foundations in empirical process theory, and can benefit from advances in Monte Carlo methods, especially in variance reduction techniques (Singh, Kantas, Vo, Doucet, & Evans, 2007; Coquelin, Deguest, & Munos, 2009; Hoffman, Kueck, Doucet, & de Freitas, 2009).

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 FRS-FNRS of which he acknowledges the financial support. This work was supported in part by the IST Programme on the European Community, under the PASCAL2 Network of Excellence, IST-2007-216886. This publication only reflects the authors' views.

References

Antos, A., Munos, R., & Szepesvári, C. (2008). Fitted Q-iteration in continuous action-space MDPs. In *Advances in Neural Information Processing Systems 20 (NIPS-2007)* (p. 9-16). Cambridge, MA: MIT Press.

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2007). *The traveling salesman problem: A computational study.* Princeton, NJ: Princeton University Press.

Arrow, K. J. (1958). Historical background. In K. J. Arrow, S. Karlin, & H. Scarf (Eds.), *Studies in the mathematical theory of inventory and production.* Stanford, CA: Stanford University Press.

Artzner, P., Delbaen, F., Eber, J.-M., Heath, D., & Ku, H. (2007). Coherent multiperiod risk adjusted values and Bellman's principle. *Annals of Operations Research*, *152*(1), 5-22.

Balasubramanian, J., & Grossmann, I. E. (2003). Approximation to multistage stochastic optimization in multiperiod batch plant scheduling under demand uncertainty. *Industrial & Engineering Chemistry Research*(43), 3695-3713.

Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM*, *9*, 61-63.

Ben-Tal, A., El Ghaoui, L., & Nemirovski, A. (2009). *Robust optimization.* Princeton, NJ: Princeton University Press.

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

Billingsley, P. (1995). *Probability and measure* (Third ed.). New York, NY: Wiley-Interscience.

Birge, J., & Louveaux, F. (1997). *Introduction to stochastic programming.* New York, NY: Springer.

Birge, J. R. (1992). The value of the stochastic solution in stochastic linear programs with fixed recourse. *Mathematical Programming*, *24*, 314–325.

Boda, K., & Filar, J. A. (2006). Time consistent dynamic risk measures. *Mathematical Methods of Operations Research*, *63*, 169-186.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization.* Cambridge, UK: Cambridge University Press.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees.* Boca Raton, FL: Chapman and Hall/CRC.

Carpentier, P., Cohen, G., & Culioli, J. C. (1996). Stochastic optimization of unit commitment: A new decomposition framework. *IEEE Transactions on Power Systems*, *11*, 1067-1073.

Chiralaksanakul, A. (2003). *Monte Carlo methods for multi-stage stochastic programs.* Unpublished doctoral dissertation, University of Texas at Austin, Austin, TX.

Chung, K.-J., & Sobel, M. (1987). Discounted MDP's: Distribution functions and exponential utility maximization. *SIAM Journal on Control and Optimization*, *25*(1), 49-62.

Coquelin, P.-A., Deguest, R., & Munos, R. (2009). Particle filter-based policy gradient in POMDPs. In *Advances in Neural Information Processing Systems 21 (NIPS-2008)* (p. 337-344). Cambridge, MA: MIT Press.

Csáji, B., & Monostori, L. (2008). Value function based reinforcement learning in changing Markovian environments. *Journal of Machine Learning Research*, *9*, 1679-1709.

Dantzig, G. B. (1955). Linear programming under uncertainty. *Management Science*, *1*, 197-206.

Defourny, B., Ernst, D., & Wehenkel, L. (2008, December). *Risk-aware decision making and dynamic programming.* Paper presented at the NIPS-08 workshop on model uncertainty and risk in reinforcement learning, Whistler, BC.

Defourny, B., Ernst, D., & Wehenkel, L. (2009). Bounds for multistage stochastic programs using supervised learning strategies. In *Stochastic Algorithms: Foundations and Applications. Fifth International Symposium, SAGA 2009* (p. 61-73). Berlin, Germany: Springer-Verlag.

Delage, E., & Ye, Y. (2008). Distributionally Robust Optimization under Moment Uncertainty with Application to Data-Driven Problems. (To appear in *Operations Research*)

Dempster, M. A. H., Pflug, G., & Mitra, G. (Eds.). (2008). *Quantitative fund management.* Boca Raton, FL: Chapman & Hall/CRC.

Demuth, H., & Beale, M. (1993). *Neural network toolbox for use with Matlab.*

Dentcheva, D., & Römisch, W. (2004). Duality gaps in nonconvex stochastic optimization. *Mathematical Programming*, *101*(3), 515-535.

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. In *Proceedings of the first international workshop on multiple classifier systems* (p. 1-15). Berlin, Germany: Springer-Verlag.

Epstein, L., & Schneider, M. (2003). Recursive multiple-priors. *Journal of Economic Theory*, *113*, 1-13.

Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, *6*, 503-556.

Ernst, D., Glavic, M., Capitanescu, F., & Wehenkel, L. (2009). Reinforcement learning versus model predictive control: A comparison on a power system problem. *IEEE Transactions on Systems, Man and Cybernetics - Part B: Cybernetics*, *39*(2), 517-529.

Escudero, L. F. (2009). On a mixture of the fix-and-relax coordination and Lagrangian substitution schemes for multistage stochastic mixed integer programming. *Top*, 5-29.

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, *63*.

Ghavamzadeh, M., & Engel, Y. (2007). Bayesian policy gradient algorithms. In *Advances in Neural Information Processing Systems 19 (NIPS-2006)* (p. 457-464). Cambridge, MA: MIT Press.

Goel, V., & Grossmann, I. E. (2006). A class of stochastic programs with decision dependent uncertainty. *Mathematical Programming*, *108*, 355-394.

Grant, M., & Boyd, S. (2008). Graph implementations for nonsmooth convex programs. *Recent Advances in Learning and Control – A tribute to M. Vidyasagar*, 95-110.

Grant, M., & Boyd, S. (2009, February). *CVX: Matlab software for disciplined convex programming (web page and software).* (http://stanford.edu/~boyd/cvx)

Hammond, P. J. (1976). Changing tastes and coherent dynamic choice. *The Review of Economic Studies*, *43*, 159-173.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (Second ed.). New York, NY: Springer.

Heitsch, H., & Römisch, W. (2003). Scenario reduction algorithms in stochastic programming. *Computational Optimization and Applications*, *24*, 187-206.

Heitsch, H., & Römisch, W. (2009). Scenario tree modeling for multistage stochastic programs. *Mathematical Programming*, *118*(2), 371-406.

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

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 1527-1554.

Hochreiter, R., & Pflug, G. C. (2007). Financial scenario generation for stochastic multi-stage decision processes as facility location problems. *Annals of Operations Research*, *152*, 257-272.

Hoffman, M., Kueck, H., Doucet, A., & de Freitas, N. (2009). New inference strategies for solving Markov decision processes using reversible jump MCMC. In *Proceedings of the twenty-fifth conference on Uncertainty in Artificial Intelligence (UAI-2009)* (p. 223-231). AUAI Press.

Howard, R. A., & Matheson, J. (1972). Risk-sensitive Markov Decision Processes. *Management Science*, *18*(7), 356-369.

Høyland, K., Kaut, M., & Wallace, S. W. (2003). A heuristic for moment-matching scenario generation. *Computational Optimization and Applications*, *24*, 1573-2894.

Huang, K., & Ahmed, S. (2009). The value of multistage stochastic programming in capacity planning under uncertainty. *Operations Research*, *57*, 893-904.

Infanger, G. (1992). Monte Carlo (importance) sampling within a Benders decomposition algorithm for stochastic linear programs. *Annals of Operations Research*, *39*, 69-95.

Kallrath, J., Pardalos, P. M., Rebennack, S., & Scheidt, M. (Eds.). (2009). *Optimization in the energy industry.* Berlin, Germany: Springer-Verlag.

Kearns, M. J., Mansour, Y., & Ng, A. Y. (2002). A sparse sampling algorithm for near-optimal planning in large Markov Decision Processes. *Machine Learning*, *49*(2-3), 193-208.

Koivu, M., & Pennanen, T. (2010). Galerkin methods in dynamic stochastic programming. *Optimization*, 339-354.

Kouwenberg, R. (2001). Scenario generation and stochastic programming models for asset liability management. *European Journal of Operational Research*, *134*, 279-292.

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

Kuhn, D. (2005). *Generalized bounds for convex multistage stochastic programs* (Vol. 548). Berlin, Germany: Springer-Verlag.

Kydland, F. E., & Prescott, E. C. (1977). Rules rather than discretion: The inconsistency of optimal plans. *The Journal of Political Economy*, *85*, 473-492.

Lagoudakis, M. G., & Parr, R. (2003). Reinforcement learning as classification: leveraging modern classifiers. In *Proceedings of the twentieth International Conference on Machine Learning (ICML-2003)* (p. 424-431). Menlo Park, CA: AAAI Press.

Langford, J., & Zadrozny, B. (2005). Relating reinforcement learning performance to classification performance. In *Proceedings of the twenty-second International Conference on Machine Learning (ICML-2005)* (p. 473-480). New York, NY: ACM.

Littman, M. L., Dean, T. L., & Kaelbling, L. P. (1995). On the complexity of solving Markov Decision Problems. In *Proceedings of the eleventh conference on Uncertainty in Artificial Intelligence (UAI-1995)* (p. 394-402). San Francisco, CA: Morgan Kaufmann.

MacKay, D. J. C. (2003). *Information theory, inference and learning algorithms.* Cambridge, UK: Cambridge University Press.

Mak, W.-K., Morton, D. P., & Wood, R. K. (1999). Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters*, *24*(1-2), 47-56.

Mercier, L., & Van Hentenryck, P. (2007). Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *Proceedings of the twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)* (p. 1979-1984). San Francisco, CA: Morgan Kaufmann.

Munos, R., & Szepesvári, C. (2008). Finite-time bound for fitted value iteration. *Journal of Machine Learning Research*, *9*, 815-857.

Nemirovski, A., Juditsky, A., Lan, G., & Shapiro, A. (2009). Stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, *19*, 1574-1609.

Nesterov, Y. (2003). *Introductory lectures on convex optimization.* Dordrecht, The Netherlands: Kluwer Academic Publishers.

Ng, A. Y., & Jordan, M. (1999). PEGASUS: a policy search method for large MDPs and POMDPs. In *Proceedings of the sixteenth conference on Uncertainty in Artificial Intelligence (UAI-2000)* (p. 406-415). San Francisco, CA: Morgan Kaufmann.

Norkin, V. I., Ermoliev, Y. M., & Ruszczyński, A. (1998). On optimal allocation of indivisibles under uncertainty. *Operations Research*, *46*, 381-395.

Pages, G., & Printems, J. (2003). Optimal quadratic quantization for numerics: the Gaussian case. *Monte Carlo Methods and Applications*, *9*, 135-166.

Pennanen, T. (2009). Epi-convergent discretizations of multistage stochastic programs via integration quadratures. *Mathematical Programming*, *116*, 461-479.

Pflug, G. C., & Römisch, W. (2007). *Modeling, measuring and managing risk.* Hackensack, NJ: World Scientific Publishing Company.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality.* Hoboken, NJ: Wiley-Interscience.

Powell, W. B., & Topaloglu, H. (2003). Stochastic programming in transportation and logistics. In A. Ruszczyński & A. Shapiro (Eds.), *Stochastic Programming.* Handbooks in Operations Research and Management Science (Vol. 10, p. 555-635). Amsterdam, The Netherlands: Elsevier.

Prékopa, A. (1995). *Stochastic programming.* Dordrecht, The Netherlands: Kluwer Academic Publishers.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete stochastic dynamic programming.* Hoboken, NJ: Wiley.

Rachev, S. T., & Römisch, W. (2002). Quantitative stability in stochastic programming: The method of probability metrics. *Mathematical Programming*, *27*(4), 792-818.

Raiffa, H., & Schlaifer, R. (1961). *Applied statistical decision theory.* Cambridge, MA: Harvard University Press.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning.* Cambridge, MA: MIT Press.

Rockafellar, R. T. (1970). *Convex analysis.* Princeton, NJ: Princeton University Press.

Rockafellar, R. T., & Uryasev, S. (2000). Optimization of conditional value-at-risk. *Journal of Risk*, *2*(3), 21-41.

Rockafellar, R. T., & Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research*, *16*, 119-147.

Samuelson, P. A. (1937). A note on measurement of utility. *The Review of Economic Studies*, *4*(2), 155-161.

Schultz, R., Stougie, L., & Van der Vlerk, M. H. (1998). Solving stochastic programs with integer recourse by enumeration: A framework using Gröbner basis reduction. *Mathematical Programming*, *83*, 229-252.

Sen, S., Doverspike, R. D., & Cosares, S. (1994). Network planning with random demand. *Telecommunication Systems*, *3*, 11-30.

Sen, S., & Sherali, H. (2006). Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. *Mathematical Programming*, *106*, 203-223.

Sen, S., Yu, L., & Genc, T. (2006). A stochastic programming approach to power portfolio optimization. *Operations Research*, *54*, 55-72.

Shapiro, A. (2003a). Inference of statistical bounds for multistage stochastic programming problems. *Mathematical Methods of Operations Research*, *58*(1), 57-68.

Shapiro, A. (2003b). Monte Carlo sampling methods. In A. Ruszczyński & A. Shapiro (Eds.), *Stochastic Programming.* Handbooks in Operations Research and Management Science (Vol. 10, p. 353-425). Amsterdam, The Netherlands: Elsevier.

Shapiro, A. (2006). On complexity of multistage stochastic programs. *Operations Research Letters*, *34*(1), 1-8.

Shapiro, A. (2009). On a time-consistency concept in risk averse multistage stochastic programming. *Operations Research Letters*, *37*, 143-147.

Shapiro, A., Dentcheva, D., & Ruszczyński, A. (2009). *Lectures on stochastic programming: Modeling and theory.* Philadelphia, PA: SIAM.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological Review*, *63*, 129-138.

Singh, S. S., Kantas, N., Vo, B.-N., Doucet, A., & Evans, R. J. (2007). Simulation-based optimal sensor scheduling with application to observer trajectory planning. *Automatica*, *43*, 817-830.

Steinwart, I., & Christman, A. (2008). *Support Vector Machines.* New York, NY: Springer.

Strotz, R. H. (1955). Myopia and inconsistency in dynamic utility maximization. *The Review of Economic Studies*, *23*, 165-180.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning, an introduction.* Cambridge, MA: MIT Press.

The MathWorks, Inc. (2004). *Matlab.* (http://www.mathworks.com)

Van der Vlerk, M. H. (2009). Convex approximations for a class of mixed-integer recourse models. *Annals of Operations Research.* (Springer Online First)

Van Hentenryck, P., & Bent, R. (2006). *Online stochastic combinatorial optimization.* Cambridge, MA: MIT Press.

Vapnik, V. N. (1998). *Statistical learning theory.* New York, NY: John Wiley & Sons.

Verweij, B., Ahmed, S., Kleywegt, A., Nemhauser, G., & Shapiro, A. (2003). The Sample Average Approximation method applied to stochastic routing problems: A computational study. *Computational Optimization and Applications*, *24*(2-3), 289-333.

Wallace, S. W., & Ziemba, W. T. (Eds.). (2005). *Applications of stochastic programming.* Philadelphia, PA: SIAM.

Wets, R. J.-B. (1974). Stochastic programs with fixed recourse: The equivalent deterministic program. *SIAM Review*, *16*, 309-339.