Bounds for Multistage Stochastic Programs using Supervised Learning Strategies

Boris Defourny, Damien Ernst, and Louis Wehenkel

University of Liège, Systems and Modeling, B28,
B-4000 Liège, Belgium
bdf@montefiore.ulg.ac.be,
WWW home page: http://www.montefiore.ulg.ac.be/~bdf/

Abstract. We propose a generic method for obtaining quickly good upper bounds on the minimal value of a multistage stochastic program. The method is based on the simulation of a feasible decision policy, synthesized by a strategy relying on any scenario tree approximation from stochastic programming and on supervised learning techniques from machine learning.

1 Context

Let Ω denote a measurable space equipped with a sigma algebra $\mathcal{B}$ of subsets of Ω, defined as follows. For $t = 0, 1, \dots, T-1$, we let ξ_t be a random variable valued in a subset of an Euclidian space Ξ_t, and let $\mathcal{B}_t$ denote the sigma algebra generated by the collection of random variables $\xi_{[0:\, t-1]} \overset{\text{def}}{=} \{\xi_0, \dots, \xi_{t-1}\}$, with $\mathcal{B}_0 = \{\varnothing, \Omega\}$ corresponding to the trivial sigma algebra. Then we set $\mathcal{B}_{T-1} = \mathcal{B}$. Note that $\mathcal{B}_0 \subset \mathcal{B}_1 \subset \cdots \subset \mathcal{B}_{T-1}$ form a filtration; without loss of generality, we can assume that the inclusions are proper — that is, ξ_t cannot be reduced to a function of $\xi_{[0:\, t-1]}$.

Let $\pi_t : \Xi \to U_t$ denote a $\mathcal{B}_t$-measurable mapping from the product space $\Xi = \Xi_0 \times \cdots \times \Xi_{T-1}$ to an Euclidian space U_t, and let Π_t denote the class of such mappings. Of course Π_0 is a class of real-valued constant functions.

We equip the measurable space $(\Omega, \mathcal{B})$ with the probability measure $\mathbb{P}$ and consider the following optimization program, which is a *multistage stochastic program* put in abstract form:

$$\mathcal{S} : \quad \min_{\pi \in \Pi} \mathbb{E}\{f(\xi, \pi(\xi))\} \qquad \text{subject to } \pi_t(\xi) \in \mathcal{U}_t(\xi) \text{ almost surely.} \tag{1}$$

Here ξ denotes the random vector $[\xi_0 \dots \xi_{T-1}]$ valued in Ξ, and π is the mapping from Ξ to the product space $U = U_0 \times \cdots \times U_{T-1}$ defined by $\pi(\xi) = [\pi_0(\xi) \ \dots \pi_{T-1}(\xi)]$ with $\pi_t \in \Pi_t$. We call such π an *implementable policy* and let Π denote the class of implementable policies.

The function $f : \Xi \times U \to \mathbb{R} \cup \{\pm\infty\}$ is $\mathcal{B}$-measurable and such that $f(\cdot, \xi)$ is convex for each ξ. It can be interpreted as a multi-period cost function.

The sets $\mathcal{U}_t(\xi)$ are defined as $\mathcal{B}_t$-measurable closed convex subsets of U_t. A set $\mathcal{U}_t(\xi)$ may implicitly depend on $\pi_0(\xi), \dots, \pi_{t-1}(\xi)$ viewed as $\mathcal{B}_t$-measurable random variables; in that context we let $\mathcal{U}_0$ be nonempty, and $\mathcal{U}_t(\xi)$ be nonempty whenever $\pi_\tau(\xi) \in \mathcal{U}_\tau(\xi)$ for each $\tau < t$ — these conditions correspond to a *relatively complete recourse* assumption.

The interpretation of the measurability restrictions put on the class Π is that the sequence $\{u_t\} \overset{\text{def}}{=} \{\pi_t(\xi)\}$ depends on information on ξ gradually revealed. Note the impact of these restrictions: should the mappings π_t be $\mathcal{B}$-measurable, and then an optimal solution would simply be given by $u = \pi(\xi)$ with $u = [u_0 \ \dots u_{T-1}]$ a real vector minimizing $f(\xi, u)$ subject to constraints $u_t \in \mathcal{U}_t(\xi)$ and optimized separately for each ξ.

A usual approximation for estimating the optimal value of programs of the form (1) consists in replacing the probability space $(\Omega, \mathcal{B}, \mathbb{P})$ by a simpler one, so as to replace the expectation by a finite sum and make the optimization program tractable. One then ends up with a finite number of possible realizations $\xi^{(k)}$ of ξ — called *scenarios* — and a probability measure $\mathbb{P}'$ induced by probability masses $p^{(k)} = \mathbb{P}'(\xi = \xi^{(k)}) > 0$ satisfying $\sum_{k=1}^n p^{(k)} = 1$. The program (1) then becomes

$$\mathcal{S}' : \quad \min_{\pi \in \Pi'} \sum_{k=1}^{n} p^{(k)} f(\xi^{(k)}, \pi(\xi^{(k)})) \qquad \text{subject to } \pi_t(\xi^{(k)}) \in \mathcal{U}_t(\xi^{(k)}) \ \forall k \ . \quad (2)$$

It has a finite number of optimization variables $u_t^{(k)} \overset{\text{def}}{=} \pi_t(\xi^{(k)})$. Remark that the random variables ξ_t have actually been replaced by other random variables ξ'_t, and the sigma subalgebras $\mathcal{B}_t$ actually replaced by those $\mathcal{B}'_t$ generated by the collection $\xi'_{[0:\, t-1]} = \{\xi'_0, \dots, \xi'_{t-1}\}$. In particular, $\mathcal{B}$ is now identified to $\mathcal{B}'_{T-1}$, and the class Π'_t is such that π_t is $\mathcal{B}'_t$-measurable.

This paper is interested in the following question. We want to **generalize** a solution for (2) — that is, values of $\pi_t(\xi)$ on a finite subset of points of Ξ — to a good candidate solution $\hat{\pi}$ for (1), of course in such a way that $\hat{\pi} \in \Pi$ and $\hat{\pi}_t(\xi) \in \mathcal{U}_t$ (**implementability** and **feasibility**). Moreover, we want that $\hat{\pi}(\xi)$ be easy to evaluate (**low complexity**), so that we can, considering a test sample TS of m independent and identically distributed (i.i.d.) new scenarios $\xi^{(j)}$, compute efficiently an unbiased estimator of $\mathbb{E}\{f(\xi, \hat{\pi}(\xi))\}$ by

$$\mathcal{R}_{\text{TS}}(\hat{\pi}) = \frac{1}{m} \sum_{j=1}^{m} f(\xi^{(j)}, \hat{\pi}(\xi^{(j)})) \ , \quad (3)$$

which provides, up to the standard error of (3), and given that $\hat{\pi}$ is feasible but possibly suboptimal, an upper bound on the optimal value of (1).

The rest of the paper is organized as follows. Section 2 motivates the question. Section 3 describes properties of the data extracted from approximate programs (2). Section 4 casts the generalization problem within a standard supervised learning framework and discusses the possible adaptations to the present context. Section 5 proposes learning algorithms. Section 6 points to related work and Section 7 concludes.

2 Motivation

Coming back to the transition from (1) to (2), we observe that there exist, in the stochastic programming literature, algorithms $\mathcal{A}(\Theta)$, with Θ any algorithm-dependent parameters, that take as input $(\Omega, \mathcal{B}, \mathbb{P})$, possibly under the form of the conditional distributions of ξ_t given $\xi_{[0:\,t-1]}$, and return an approximation $(\Omega', \mathcal{B}', \mathbb{P}')$ under the form of n scenario-probability pairs. These pairs are generally arranged under the form of a *scenario tree* [1–9], from which a version of (2) can be formulated and solved. But this approach raises several concerns, in particular on the good choices for the parameters Θ.

- It has been shown in [10] for a particular randomized algorithm $\mathcal{A}_1(\Theta_1)$ relying on Monte Carlo sampling — the Sample Average Approximation (SAA) [5] — that it is necessary, for guaranteeing with a certain probability $1-\alpha$ that an optimal value of (2) is ϵ-close to that of (1), to have a number of scenarios n that grows with T exponentially. Thus in practice, one might have to use unreliable approximations built with a small fraction of this number n. What happens then is not clear. It has also been shown in [11] that it is not possible to obtain an upper bound on the optimal value of (1) using statistical estimates based on $\mathcal{A}_1(\Theta_1)$ when $T > 2$.
- It has been shown in [8] that there exists a particular derandomized algorithm $\mathcal{A}_2(\Theta_2)$ relying on Quasi-Monte Carlo sampling and valid for a certain class of models for ξ, such that the optimal value and solution set of (2) converge to those of (1) asymptotically. What happens in the non-asymptotical case is not clear.
- It has been shown in [9] that there exist two polynomial-time deterministic heuristics $\mathcal{A}_3(\Theta_3)$ — the backward and forward tree constructions — that approximate the solution to a combinatorial problem, ideal in a certain sense, which is NP-hard. However, the choice of Θ_3 is left as largely open.

There actually exists a generic method for computing upper bounds on the value of (1) using any particular algorithm $\mathcal{A}(\Theta)$. This method has been used by authors studying the value of stochastic programming models [12] or the performance of particular algorithms $\mathcal{A}(\Theta)$ [13, 14]. It relies on an estimate of the form (3) with $\hat{\pi}$ replaced by a certain policy $\tilde{\pi}$ defined implicitly. It has computational requirements which turn out to be **extreme**. To see this, we describe the procedure used to compute the sequence $\{\tilde{\pi}_t(\xi^{(j)})\}$ corresponding to a particular scenario $\xi^{(j)}$.

Assuming that the solution to the $\mathcal{A}(\Theta)$-approximation (2) *of a problem $\mathcal{S}$ of the form* (1) *is available, in particular $u_0^{(1)} \in \Pi_0 \cap \mathcal{U}_0$, one begins by setting $\tilde{\pi}_0(\xi^{(j)}) = u_0^{(1)}$. Then, one considers a new problem, say $\mathcal{S}(\xi_0^{(j)})$, derived from $\mathcal{S}$ by replacing the random variable ξ_0 by its realization $\xi_0^{(j)}$ — and possibly taking account of $\tilde{\pi}_0(\xi^{(j)})$ for simplifying the formulation. This makes the sigma algebra $\mathcal{B}_1$ in* (1) *trivial and Π_1 a class of constant functions. One uses again the algorithm $\mathcal{A}(\Theta)$, possibly with an updated Θ, to obtain an approximate version* (2) *of $\mathcal{S}(\xi_0^{(j)})$. One computes its solution, extracts $u_1^{(1)} \in \Pi_1 \cap \mathcal{U}_1$ from* *it, and sets $\tilde{\pi}_1(\xi^{(j)}) = u_1^{(1)}$. The procedure is repeated for each $t = 2, \dots, T-1$, on the sequence of problems $\mathcal{S}(\xi^{(j)}_{[0:\,t-1]})$ formed by using the growing collection of observations $\xi^{(j)}_{[0:\,t-1]}$ extracted from $\xi^{(j)}$.*

It is not hard to see that the resulting sequence $\tilde{\pi}_t(\xi^{(j)})$ forming $\tilde{\pi}(\xi^{(j)})$ meets the implementability and feasibility conditions. But repeating the full evaluation process on a sufficient number m of scenarios is particularly cumbersome. The bottleneck may come from the complexity of the programs to solve, from the running time of the algorithm $\mathcal{A}(\Theta)$ itself, or, if $\mathcal{A}(\Theta)$ is a randomized algorithm, from the new source of variance it adds to the Monte-Carlo estimate (3).

Now an upper bound on the value of (1) computed quickly and reflecting the quality of $\mathcal{A}(\Theta)$ would allow a more systematic exploration of the space of parameters Θ. In the case of a randomized algorithm such as $\mathcal{A}_1(\Theta_1)$, it may allow to rank candidate solutions obtained through several runs of the algorithm.

3 Datasets

We consider a problem $\mathcal{S}$ and the approximate version (2) obtained through an algorithm $\mathcal{A}(\theta)$. Let $(p^{(i)}, \xi^{(i)}, u^{(i)})$ denote a probability-scenario-decision triplet formed by extracting from an optimal solution to (2) the sequence $u^{(i)} = [u_0^{(i)} \ \dots \ u_{T-1}^{(i)}]$ that corresponds to the scenario $\xi^{(i)} = [\xi_0^{(i)} \ \dots \ \xi_{T-1}^{(i)}]$ of probability $p^{(i)}$. There are n of such triplets; let us collect them into a dataset $\mathcal{D}_n$.

Definition 1 (Regular datasets).
In this paper, a dataset is said to be $\mathcal{S}-$regular*, or simply* regular*, if the decisions $u^{(i)}$ of its triplets $(p^{(i)}, \xi^{(i)}, u^{(i)})$ are jointly optimal with respect to a program* (2) *over the complete set of probability-scenario pairs $(p^{(i)}, \xi^{(i)})$.*

Normal datasets have two immediate but important properties.

Proposition 1 (Properties of regular datasets).

1. *For any pair i, j of triplets $(p^{(i)}, \xi^{(i)}, u^{(i)})$ and $(p^{(j)}, \xi^{(j)}, u^{(j)})$ in $\mathcal{D}_n$, the* non-anticipativity *property holds: for $t = 0$, $u_t^{(i)} = u_t^{(j)}$ and for $1 \le t \le \ T-1$, if $\xi^{(i)}_{[0:\,t-1]} = \xi^{(j)}_{[0:\,t-1]}$ then $u_t^{(i)} = u_t^{(j)}$.*
2. *For any triplet $(p^{(i)}, \xi^{(i)}, u^{(i)})$, the* feasibility *property holds: for all $t \ge 0$, $u_t^{(i)} \in \mathcal{U}_t(\xi^{(i)})$.*

The interpretation of these properties may be better perceived through the proof.

Proof. Property 1 results from the measurability restriction put on the classes Π_t. Since the sigma algebras $\mathcal{B}_t$ are generated by the random variables $\xi_{[0:\,t-1]}$, the decisions $u_t = \pi_t(\xi)$ are $\mathcal{B}_t$-measurable if and only if there exists a measurable map $g_t : \Xi_0 \times \dots \times \Xi_{t-1} \to U_t$ such that $u_t = g_t(\xi_0, \dots, \xi_{t-1})$ in any event (Theorem 20.1 in [15]). The latter set of conditions is enforced in (2) either by imposing explicit equality constraints $u_t^{(i)} = u_t^{(j)}$ whenever $\xi^{(i)}_{[0:\,t-1]} = \xi^{(j)}_{[0:\,t-1]}$, or by merging those equal optimization variables into a single one.
Property 2 is directly enforced by the constraint $u_t^{(i)} \overset{\text{def}}{=} \pi_t(\xi^{(i)}) \in \mathcal{U}_t(\xi^{(i)})$ in (2).

Definition 2 (NAF property). *In this paper, a dataset is said to have the NAF property if its triplets mutually satisfy the N̲on-A̲nticipativity properties, and if each triplet satisfies the F̲easibility properties relative to $\mathcal{S}$.*

Remark 1. Let $\mathcal{D}_{n+1}$ be a regular dataset with triplets having the same probability $p^{(i)} = (n+1)^{-1}$. Let $\mathcal{D}_n$ be the dataset obtained by deleting one triplet from $\mathcal{D}_{n+1}$ and setting $p^{(i)} = n^{-1}$ for each remaining triplet. Then $\mathcal{D}_n$ has still the NAF property but is not necessarily regular.

Any dataset with the NAF property induces a set of *compatible policies*, in the following sense.

Proposition 2 (Compatible policies).
Let $\mathcal{D}_n$ be a dataset of triplets $(p^{(k)}, \xi^{(k)}, u^{(k)})$ with the NAF property. Let $\mathcal{P}(\mathcal{S})$ denote the set of implementable and feasible policies for $\mathcal{S}$. We assume that $\mathcal{P}(\mathcal{S})$ is nonempty. We define $\mathcal{P}(\mathcal{D}_n)$ as the set of policies $\pi \in \Pi$ such that $\pi(\xi^{(k)}) = u^{(k)}$, $1 \le k \le n$. Then it holds that $\mathcal{P}(\mathcal{S}) \cap \mathcal{P}(\mathcal{D}_n)$ is nonempty. In particular, each policy π in $\mathcal{P}(\mathcal{S}) \cap \mathcal{P}(\mathcal{D}_n)$ satisfies $\pi_0(\xi) = u_0^{(1)}$, and $\pi_t(\xi) = u_t^{(i)}$ whenever $\xi_{[0:\, t-1]} = \xi^{(i)}_{[0:\, t-1]}$ for some $1 \le i \le n$ and $0 < t < T-1$. These policies are said to be compatible *with the dataset.*

Proof. We recall that $\pi \in \mathcal{P}(\mathcal{S})$ iff $\pi \in \Pi$ and $\pi_t(\xi) \in \mathcal{U}_t(\xi)$ almost surely. If there exists a policy π in $\mathcal{P}(\mathcal{S}) \cap \mathcal{P}(\mathcal{D}_n)$, it is clear, by the feasibility property of the triplets in $\mathcal{D}_n$, that $\pi_t(\xi^{(k)}) = u_t^{(k)}$ is in $\mathcal{U}_t(\xi^{(k)})$ for each k, t. The properties stated in the proposition then follow from the $\mathcal{B}_t$-measurability of π_t. It remains to show that given the value of the mapping π over the points $\Xi^{\mathcal{D}_n} \overset{\text{def}}{=} \{\xi^{(1)}, \ldots, \xi^{(n)}\}$, it is still possible to specify $\pi(\xi)$ for any $\xi \in \Xi \setminus \Xi^{\mathcal{D}_n}$ such that $\pi_t(\xi) \in \mathcal{U}_t(\xi)$. But this follows from the relatively complete recourse assumption: for any scenario ξ such that $\xi_{[0:\, t-1]} = \xi^{(i)}_{[0:\, t-1]}$ for some i, t, setting $\pi_\tau(\xi) = u_\tau^{(i)}$ for each $\tau \le t$ cannot make the sets $\mathcal{U}_{t+1}(\xi), \ldots, \mathcal{U}_{T-1}(\xi)$ empty.

4 Generalization Strategies

Proposition 2 suggests to view each pair $(\xi^{(k)}, u^{(k)})$ of a dataset $\mathcal{D}_n$ with the NAF property as being generated by a single implementable and feasible policy $\pi \in \mathcal{P}(\mathcal{D}_n)$. One can thus try to predict the output $\pi(\xi)$ of that policy on any new scenario, on the basis of the triplets of $\mathcal{D}_n$. However, the problem is not well-posed if we cannot discriminate among the policies compatible with the dataset. There exists two classical approaches to further specify the problem:

1. Choose a class of models on which the search for π can be restricted;
2. Put a prior probability over every possible function for π, compute posterior probabilities using the observations provided by the dataset, and then make a prediction for the policy.

It turns out that the two approaches raise similar issues regarding their adaptation to the context of this paper; we pursue the discussion using the first one.

Definition 3 (Training sets).
Starting from the triplets of a dataset $\mathcal{D}_n$, for each $t > 0$, let S_t be a set made of the pairs $(x^{(k)}, y^{(k)})$ with $x^{(k)} \overset{def}{=} (\xi_0^{(k)}, \ldots, \xi_{t-1}^{(k)})$ and $y^{(k)} \overset{def}{=} u_t^{(k)}$.

Definition 4 (Supervised Learning — Hypothesis Class View).
Consider a training set S_t of m pairs $(x^{(k)}, y^{(k)}) \in X_t \times Y_t$. Assume that these pairs are i.i.d. samples drawn from a fixed but unknown distribution P_t over $X_t \times Y_t$. Consider a hypothesis class $\mathcal{H}_t$ of mappings $h_t : X_t \to Y_t$. The goal of a supervised learning algorithm $\mathcal{L} : [X_t \times Y_t]^m \to \mathcal{H}_t$ is to select a hypothesis $h_t \in \mathcal{H}_t$ such that $h_t(x)$ approximates y in the best possible way on new samples (x, y) drawn from the distribution P_t.

In many approaches to supervised learning, the selection of the hypothesis h_t is done by first specifying a *loss function* $\ell : X \times Y \times Y \to \mathbb{R}^+$ for penalizing the error between the prediction $h_t(x)$ and the target y, and a complexity measure $C : \mathcal{H} \to \mathbb{R}^+$ for penalizing the complexity of the hypothesis [16, 17]. Defining the *empirical risk* of a hypothesis $h_t \in \mathcal{H}_t$ as

$$\mathcal{R}_{\ell,S_t}(h_t) = m^{-1} \sum_{k=1}^{m} \ell(x^{(k)}, y^{(k)}, h_t(x^{(k)})) \ ,$$

one selects the hypothesis h_t by minimizing a weighted sum of the empirical risk and the complexity measure (we use $\lambda \geq 0$):

$$h_t^*(\lambda) = \arg \min_{h_t \in \mathcal{H}_t} \mathcal{R}_{\ell,S_t}(h_t) + \lambda\, C(h_t) \ . \tag{4}$$

Choosing the right value for λ is essentially a *model selection* issue. One seeks to choose λ such that $h_t^*(\lambda)$ would also minimize the *generalization error*

$$\mathcal{R}_\ell(h_t) = \mathbb{E}_{(x,y)\sim P_t}\{\ell(x, y, h_t(x))\} \ . \tag{5}$$

Ideally, the dataset is large, and we can partition the samples into a training set, a validation set, and a test set. Hypotheses $h_t^*(\lambda)$ are optimized on the training set, and then ranked on the basis of their risk on the validation set. The best hypothesis $h_t^*(\lambda^*)$ is retained and its generalization error is estimated on the test set. Otherwise, when the dataset is small, there exist computationally-intensive techniques to estimate the generalization error by working on several random partitions of the dataset [18, 19].

To actually produce an upper bound in (3), the prediction of the output of $\pi(\xi)$ must be implementable and feasible. While it is not hard to decompose the learning of π into a sequence of learning tasks for $\pi_0, \pi_1(\xi_1), \ldots, \pi_{T-1}(\xi_{[0:\, T-2]})$, more care is needed for enforcing the feasibility of a prediction.

Let $\Xi_{[0:\, t]}$ be a shorthand for $\Xi_0 \times \cdots \times \Xi_t$ and $U_{[0:\, t]}$ be a shorthand for $U_0 \times \cdots \times U_t$.

Definition 5 (Feasibility mappings).
We call $M_t : \Xi_{[0:\, t-1]} \times U_{[0:\, t]} \to U_t$ a $\mathcal{U}_t$-feasibility mapping if its range is always contained in the feasible set $\mathcal{U}_t(\xi)$ whenever $\mathcal{U}_t(\xi)$ is nonempty.

Example 1 (Projections).
Let $||\cdot||$ denote a strictly convex norm on U_t. For the sake of concreteness, let $\mathcal{U}_t(\xi)$ be represented as $\{u_t \in U_t : g_k(\xi_0, \ldots, \xi_{t-1}, u_0, \ldots, u_t) \leq 0,\ 1 \leq k \leq m\}$ with g_k jointly convex in $u_0, \ldots, u_t$. The mapping M_t defined by

$$\begin{aligned} M_t(\xi_{[0:\,t-1]}, u_{[0:\,t]}) = &\arg\min_{z \in U_t} \ ||z - u_t|| \\ &\text{subject to} \quad g_k(\xi_0, \ldots, \xi_{t-1}, u_0, \ldots, u_{t-1}, z) \leq 0 \ , \quad 1 \leq k \leq m \end{aligned}$$

is a $\mathcal{U}_t$-feasibility mapping.

Proposition 3 (Representation of implementable and feasible policies).

Let M_t be feasibility mappings for the sets $\mathcal{U}_t$, $t = 0, \ldots, T-1$. Let h_0 denote a constant, and let $h_t : \Xi_{[0:\,t-1]} \to U_t$ be some mappings, $t = 1, \ldots, T-1$. Then the mapping $\hat{\pi} : \Xi \to U$ defined by

$$\begin{aligned} &\hat{\pi}(\xi) = [\hat{\pi}_0, \ldots, \hat{\pi}_{T-1}] \ , \\ &\quad \hat{\pi}_0 = M_0(h_0) \ , \qquad \hat{\pi}_t = M_t(\xi_0, \ldots, \xi_{t-1}, \hat{\pi}_0, \hat{\pi}_1, \ldots, \hat{\pi}_{t-1}, h_t(\xi_0, \ldots, \xi_{t-1})) \end{aligned}$$

corresponds to an implementable and feasible policy $\hat{\pi}$ for $\mathcal{S}$. It can be used in (3). *The computational complexity of $\hat{\pi}_t$ depends on the complexity of evaluating h_t and then M_t.*

The mappings h_t of Prop. 3 can be defined as the best hypothesis of a standard supervised learning algorithm for the learning set S_t. The model selection can be performed in two ways (referred in the sequel as M1 and M2):

1. We follow the classical approach of supervised learning outlined above, and keep some triplets of the dataset $\mathcal{D}_n$ apart for testing purpose;
2. Instead of using the loss function of the supervised learning algorithm in (5), we look for good choices of the parameters λ of the learning problems by directly computing (3) for each candidate set of parameters. *(It is also possible to perform the model selection on* (3) *on a large validation set of new scenarios, and then recompute* (3) *on a large test set of new scenarios.)*

Remark 2. It is possible to combine the two model selection approaches in a same policy $\hat{\pi}$. More generally, it is also possible to form $\hat{\pi}$ by combining learned models $\hat{\pi}_t$ on $t = 0, 1, \ldots, t_0$, and implementing on $t = t_0 + 1, \ldots, T-1$ the models $\tilde{\pi}_t$ of Section 2 based on standard stochastic programming. The two classes of models are complementary: as t increases, the learning problems for h_t become harder (larger input space, fewer samples), but the optimization problems $\mathcal{S}(\xi_{[0:t-1]})$ for $\tilde{\pi}_t$ become easier.

Up to now, we have viewed the generalization problem as the one of selecting a good model π that explains the generation of a regular dataset $\mathcal{D}_n$, and yields good predictions on new scenarios ξ in terms of a loss function from supervised learning (if we know the desired output) or in terms of the objective function of the problem (1).

Nevertheless, at the same time there is no guarantee that any policy π compatible with a dataset $\mathcal{D}_n$ having the NAF property was optimal for the original program $\mathcal{S}$. We could then interpret $u^{(k)}$ as a noisy observation of the output $\pi^*(\xi^{(k)})$ of an optimal policy π^* for $\mathcal{S}$. In this respect, we could still use the supervised learning approach of Prop. 3. We could even legitimately try to work with learning sets S_t formed from a dataset without the NAF property, for example a dataset obtained by merging the solution to several approximations (2).

We can also view the generalization problem differently. Proper algorithms $\mathcal{A}(\theta)$ come with the guarantee that the solution set of (2) converges to the solution set of (1). Thus, we could consider *two* regular datasets $\mathcal{D}_n$ and $\mathcal{D}'_q$, with $n < q$, and learn hypotheses h_t from $\mathcal{D}_n$ that generalize well to the examples from $\mathcal{D}'_q$. We will refer to this generalization approach as M3.

5 Learning Algorithms

In this section, we develop algorithms for learning the hypotheses $h_t \in \mathcal{H}_t$ of Prop. 3. We start by noting that if $u_t \in U_t$ is vector-valued with components $[u_t]_1, \dots, [u_t]_d$, the hypothesis h_t must be vector-valued as well. Of course,

Proposition 4. *Any standard regression algorithm suitable for one-dimensional outputs can be extended to multi-dimensional outputs.*

Proof. A training set $S_t = \{(x^{(i)}, y^{(i)})\}_{1\leq i\leq n}$ can be split into d training sets $[S_t]_j = \{(x^{(i)}, [y^{(i)}]_j)\}_{1\leq i\leq n}$, $j = 1, \dots, d$, from which hypotheses $[h_t]_1, \dots, [h_t]_d$ can be learned. The components $[h_t]_j$ can then be concatenated into a single vector-valued hypothesis h_t.

The following algorithm implements the approach M2 of Section 4.

Algorithm 1 (Joint selection approach).
Let $\mathcal{S}'_0$ be an approximation of the form (2) to a problem (1). Let S_t denote the training sets built from a solution to $\mathcal{S}'_0$. Assume that the model selection in (4) is parameterized by a single parameter λ common to every component j of h_t and every index t. Assume that the feasibility mappings M_t are fixed a priori. Then, for each λ in some finite set Λ,

1. Learn $[h_t]_j(\lambda)$ using training sets $[S_t]_j$ and the λ-dependent criterion (4).
2. From $[h_t]_j(\lambda)$ form a policy $\hat{\pi}(\lambda)$ (Proposition 3).
3. Evaluate the score of $\hat{\pi}(\lambda)$ with (3) on m scenarios. Let $v(\lambda)$ be that score.

Select $\lambda^* = \arg\min_{\lambda\in\Lambda} v(\lambda)$ and return the hypotheses $h_0(\lambda^*), \dots, h_{T-1}(\lambda^*)$.

Remark 3. For datasets with the NAF property, $h_0(\lambda^*) = M_0(h_0(\lambda^*)) = u_0^{(1)}$.

The purpose of the assumptions of Algorithm 1 is to reduce the complexity of the search in the parameter space Λ. To see this, let $c_{\mathcal{A}}(\Theta_0)$ denote the expected time for forming the approximation $\mathcal{S}_0'$ to $\mathcal{S}$ using algorithm $\mathcal{A}(\Theta_0)$, and let $c_S(0)$ denote the expected time for solving $\mathcal{S}_0'$. Let $|\Lambda|$ denote the cardinality of Λ, $c_L(t)$ the expected running time of the learning algorithm on a dataset S_t, and $c_E(t)$ the expected running time of the combined computation of M_t and h_t.

Proposition 5. *Algorithm 1 runs in expected time*

$$|\Lambda| \cdot \left[\sum_{t=1}^{T-1} c_L(t) + m \cdot \sum_{t=1}^{T-1} c_E(t) \right] = \sum_{t=1}^{T-1} |\Lambda| \cdot [c_L(t) + m\, c_E(t)] \quad ,$$

starting from data obtained in expected time $c_{\mathcal{A}}(\Theta_0) + c_S(0)$.

In general $c_L(t)$ and $c_E(t)$ grow with the dimensions of $\Xi_{0:t-1}$ and U_t, and with the cardinality of the dataset S_t. These dependences, and the ratio between $c_L(t)$ and $c_E(t)$, depend largely on the algorithms associated with the hypothesis spaces $\mathcal{H}_t$ and the feasibility mappings M_t. The value of $\lambda \in \Lambda$ may actually also influence $c_L(t)$ and $c_E(t)$ but we neglect this dependence in first approximation. Clearly Algorithm 1 can be run on N parallel processes, so as to replace $|\Lambda|$ in Prop. 5 by $|\Lambda|/N$.

Relaxing in Algorithm 1 the assumption of a single λ and considering parameters $\lambda_t \in \Lambda_t$ proper to each t could improve the quality of the learned policy $\hat{\pi}$ but would also imply an expected running time proportional to $\prod_{t=1}^{T-1} |\Lambda_t|$ instead of $|\Lambda|$ in Prop. 5, if one attempts a systematic search over all possible values of $[\lambda_1 \dots \lambda_{T-1}]$ in $\Lambda_1 \times \cdots \times \Lambda_{T-1}$. In that context it might be better to switch to another model selection strategy (compare to remark 2 and M3):

Algorithm 2 (Sequential selection approach).
Let $\mathcal{S}_0', \dots, \mathcal{S}_{T-1}'$ be T approximations of the form (2) to a problem (1), with $\mathcal{S}_0'$ being the only solved one.

1. From the solution to $\mathcal{S}_0'$, extract $h_0 = u_0^{(1)}$ and the training set S_1. Set $t = 1$.
2. Assume that the model selection for h_t is parameterized by a single λ_t common to every component $[h_t]_j$ of h_t. For each value of λ_t in a finite set Λ_t, learn from the training set S_t the hypothesis $h_t(\lambda_t)$.
3. Consider the optimal values $v(\lambda_t)$ of programs $\mathcal{S}_t'(\lambda_t)$ derived from $\mathcal{S}_t'$ as follows. For each $\tau = 1, \dots, t$ and each realization $\xi_{[0:\,\tau-1]}^{(k)}$ in $\mathcal{S}_t'$, compute successively, using $\hat{u}_0^{(k)} \overset{\text{def}}{=} h_0$,

   $$\hat{u}_\tau^{(k)} \overset{\text{def}}{=} M_t(\xi_0^{(k)}, \dots, \xi_{\tau-1}^{(k)}, \hat{u}_0^{(k)}, \dots, \hat{u}_{\tau-1}^{(k)}, h_\tau(\xi_0^{(k)}, \dots, \xi_{\tau-1}^{(k)}))$$

   where h_τ depends on λ_t when $\tau = t$. Then define $\mathcal{S}_t'(\lambda_t)$ as the program $\mathcal{S}_t'$ to which is added the whole set of equality constraints $u_\tau^{(k)} = \hat{u}_\tau^{(k)}$.
4. Select $\lambda_t^* = \arg\min_{\lambda_t \in \Lambda_t} v(\lambda_t)$ and set $h_t = h_t(\lambda_t^*)$.
5. If $t < T-1$, extract from the solution to $\mathcal{S}_t'(\lambda_t^*)$ the training set S_{t+1}, set t to $t+1$, and go to step 2.
   Otherwise, return the hypotheses $h_0(\lambda_0^*), \dots, h_{T-1}(\lambda_{T-1}^*)$.

Given $\hat{\pi}_0, \ldots, \hat{\pi}_{t-1}$, the impact of $\hat{\pi}_t$ on the generalization abilities of $\hat{\pi}$ is evaluated on an approximation $\mathcal{S}'_t$ to $\mathcal{S}$ distinct from those on which $\hat{\pi}_1, \ldots, \hat{\pi}_{t-1}$ have been determined. The program $\mathcal{S}'_t(\lambda_t)$ acts as a proxy for a test, on an independent set of scenarios, of a hybrid policy $\pi^\dagger$ such that (i) $\pi^\dagger_\tau = \hat{\pi}_\tau$ for $\tau \leq t$, with $\hat{\pi}_t$ depending on λ_t, and (ii) $\pi^\dagger_\tau = \pi^*_\tau$ for $\tau > t$ with π^* an optimal policy for $\mathcal{S}$. Optimizing $\mathcal{S}'_t(\lambda_t)$ yields the decisions of a policy for $u_{t+1}, \ldots, u_{T-1}$ perfectly adapted to the scenarios of $\mathcal{S}'_t$ — a source of bias with respect to π^* — and to the possibly suboptimal decisions $u_0, \ldots, u_t$ imposed on these scenarios by $\hat{\pi}_0, \ldots, \hat{\pi}_t$. Whether $v(\lambda_t)$ is lower or higher than the value of $\pi^\dagger$ on a large test sample cannot be predicted, but the *ranking* among the values $v(\lambda_t)$ should be fairly representative of the relative merits of the various models for $\hat{\pi}_t$.

The complexity of Algorithm 2 is expressed as follows. Let $c_{\mathcal{A}}(\Theta_j)$ denote the expected time for forming the approximation $\mathcal{S}'_j$ to $\mathcal{S}$ using algorithm $\mathcal{A}(\Theta_j)$. Let $m(t,\tau)$ denote the number of distinct realizations $\xi^{(k)}_{[0:\,\tau-1]}$ of $\xi_{[0:\,\tau-1]}$ in $\mathcal{S}'_t$. Let $c_{S'}(t)$ denote the expected running time for solving a program $\mathcal{S}'_t(\lambda_t)$.

Proposition 6. *Algorithm 2 runs in expected time*

$$\sum_{t=1}^{T-1} |\Lambda_t| \cdot \left[c_L(t) + \left(\sum_{\tau=1}^{t} m(t,\tau)\, c_E(\tau) \right) + c_{S'}(t) \right] \quad ,$$

starting from data obtained in expected time $\left[\sum_{j=0}^{T-1} c_{\mathcal{A}}(\Theta_j)\right] + c_S(0)$.

The term $c_{S'}(t)$ is usually large but decreases with t, since the step 3 of Algorithm 2 sets decisions up to time t and thus reduces $\mathcal{S}'_t(\lambda_t)$ to $m(t,t)$ problems over disjoint subsets of the optimization variables $u^{(k)}_{t+1}, \ldots, u^{(k)}_{T-1}$ of $\mathcal{S}'_t$. The values $m(t,\tau)$ are typically far smaller than m in Prop. 5. The term $c_{S'}(t)$ is thus essentially the price paid for estimating the generalization abilities of a policy $\hat{\pi}$ that leaves $\hat{\pi}_{t+1}, \ldots, \hat{\pi}_{T-1}$ still unspecified.

For the sake of comparison, we express the expected running time of the generic upper bounding technique of Section 2. Let $c_S(t)$ denote the expected time for solving the approximation to the program $\mathcal{S}(\xi^{(j)}_{[0:\,t-1]})$ of Section 2 obtained through $\mathcal{A}(\Theta_t)$. Let $c_{\mathcal{A}}(\Theta_t)$ denote the expected running time of the algorithm $\mathcal{A}(\Theta_t)$. Note that here Θ_t for $t > 0$ is such that there is in $\mathcal{S}(\xi^{(j)}_{[0:\,t-1]})$ a single realization of $\xi_{[0:\,t-1]}$ corresponding to the actual observation $\xi^{(j)}_{[0:\,t-1]}$ of $\xi_{[0:\,t-1]}$.

Proposition 7. *The bounding scheme of Section 2 runs in expected time*

$$\sum_{t=1}^{T-1} m \cdot [c_{\mathcal{A}}(\Theta_t) + c_S(t)] \quad ,$$

starting from data obtained in expected time $c_{\mathcal{A}}(\Theta_0) + c_S(0)$.

Of course using N parallel processes allows to replace m by m/N.

6 Related Work

The need for generalizing decisions on a subset of scenarios to an admissible policy is well recognized in the stochastic programming literature [11]. Some authors addressing this question propose to assign to a new scenario the decisions associated to the nearest scenario of the approximate solution [20, 21], thus essentially reducing the generalization problem to the one of defining a priori a measurable similarity metric in the scenario space (we note that [21] also proposes several variants for a projection step restoring the feasibility of the decisions). Put in perspective of the present framework, this amounts to adopt, without model selection, the nearest neighbor approach to regression [22] — arguably one of the most unstable prediction algorithm [17]. Our work differs in that it is concerned with the complexity of exploiting the induced policies, seeks to identify the nature of the generalization problem, and makes the connection with model selection issues well studied in statistics [23–26] and statistical learning [16, 27]. As a result, guidance is provided on how to improve over heuristical methods, and on which tools to use [28–33]. The model selection techniques developed in this paper are proper to the stochastic programming context.

There is a large body of work applying supervised learning techniques to the context of Markov Decision Processes (MDP) [34] and Reinforcement Learning problems (RL) [35, 36]. In these problems, one seeks to maximize an expected sum of rewards by finding a policy that maps states to decisions. The state space is often large, but the decision space is often reduced to a finite number of possible actions. Supervised learning is used to represent the value function of dynamic programming, or to represent the policy directly [37–41]. We observe however that the hypothesis spaces relevant for representing policies adapted to multistage stochastic programs differ from those used in the context of Markov Decision Processes, and that the datasets are obtained in a very different way.

7 Conclusion

This paper has proposed to compute bounds on the optimal value of a multistage stochastic program by generalizing to an admissible policy the solution to a scenario-tree approximation of the program. Such bounds could be helpful to select the parameters of the algorithm that generates the scenario tree, an important aspect in practical applications. We have aimed to adapt to this task the concepts and model selection techniques from machine learning and statistical learning theory. We have outlined the special structure of the data and of the output of admissible policies. The results established in this paper have also shed light on the type of feasibility constraints with which our approach can be expected to be sound. Learning algorithms for the policies have been proposed, offering various quality-complexity tradeoffs.

Acknowledgments

This paper presents research results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State, Science Policy Office. The scientific responsibility rests with its authors. Damien Ernst is a Research Associate of the Belgian FNRS of which he acknowledges the financial support.

References

1. Frauendorfer, K.: Barycentric scenario trees in convex multistage stochastic programming. Mathematical Programming **75** (1996) 277–294
2. Dempster, M.: Sequential importance sampling algorithms for dynamic stochastic programming. Annals of Operations Research **84** (1998) 153–184
3. Dupacova, J., Consigli, G., Wallace, S.: Scenarios for multistage stochastic programs. Annals of Operations Research **100** (2000) 25–53
4. Høyland, K., Wallace, S.: Generating scenario trees for multistage decision problems. Management Science **47**(2) (2001) 295–307
5. Shapiro, A.: Monte Carlo sampling methods. In Ruszczyński, A., Shapiro, A., eds.: Stochastic Programming. Handbooks in Operations Research and Management Science. Volume 10. Elsevier (2003) 353–425
6. Casey, M., Sen, S.: The scenario generation algorithm for multistage stochastic linear programming. Mathematics of Operations Research **30** (2005) 615–631
7. Hochreiter, R., Pflug, G.: Financial scenario generation for stochastic multi-stage decision processes as facility location problems. Annals of Operations Research **152** (2007) 257–272
8. Pennanen, T.: Epi-convergent discretizations of multistage stochastic programs via integration quadratures. Mathematical Programming **116** (2009) 461–479
9. Heitsch, H., Römisch, W.: Scenario tree modeling for multistage stochastic programs. Mathematical Programming **118**(2) (2009) 371–406
10. Shapiro, A.: On complexity of multistage stochastic programs. Operations Research Letters **34**(1) (2006) 1–8
11. Shapiro, A.: Inference of statistical bounds for multistage stochastic programming problems. Mathematical Methods of Operations Research **58**(1) (2003) 57–68
12. Golub, B., Holmer, M., McKendall, R., Pohlman, L., Zenios, S.: A stochastic programming model for money management. European Journal of Operational Research **85** (1995) 282–296
13. Kouwenberg, R.: Scenario generation and stochastic programming models for asset liability management. European Journal of Operational Research **134** (2001) 279–292
14. Hilli, P., Pennanen, T.: Numerical study of discretizations of multistage stochastic programs. Kybernetika **44** (2008) 185–204
15. Billingsley, P.: Probability and Measure. Third edn. Wiley (1995)
16. Vapnik, V.: Statistical Learning Theory. Wiley (1998)
17. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Second edn. Springer (2009)
18. Wahba, G., Golub, G., Heath, M.: Generalized cross-validation as a method for choosing a good ridge parameter. Technometrics **21** (1979) 215–223

19. Efron, B., Tibshirani, R.: An introduction to the bootstrap. Chapman and Hall, London (1993)
20. Thénié, J., Vial, J.P.: Step decision rules for multistage stochastic programming: A heuristic approach. Automatica **44** (2008) 1569–1584
21. Küchler, C., Vigerske, S.: Numerical evaluation of approximation methods in stochastic programming. Submitted (2008)
22. Cover, T.: Estimation by the nearest neighbor rule. IEEE Transactions on Information Theory **14** (1968) 50–55
23. Akaike, H.: Information theory and an extension of the maximum likelihood principle. In: Proceedings of the Second International Symposium on Information Theory. (1973) 267–281
24. Schwartz, G.: Estimating the dimension of a model. Annals of Statistics **6** (1978) 461–464
25. Rissanen, J.: Stochastic complexity and modeling. Annals of Statistics **14** (1986) 1080–1100
26. James, G., Radchenko, P., Lv, J.: DASSO: connections between the Dantzig selector and Lasso. Journal of the Royal Statistical Society: Series B **71** (2009) 127–142
27. Chapelle, O., Vapnik, V., Bengio, Y.: Model selection for small sample regression. Machine Learning **48** (2002) 315–333
28. Huber, P.: Projection pursuit. Annals of Statistics **13** (1985) 435–475
29. Buja, A., Hastie, T., Tibshirani, R.: Linear smoothers and additive models. Annals of Statistics **17** (1989) 453–510
30. Friedman, J.: Multivariate adaptive regression splines (with discussion). Annals of Statistics **19** (1991) 1–141
31. Girosi, F., Jones, M., Poggio, T.: Regularization theory and neural networks architectures. Neural Computation **7** (1995) 219–269
32. Williams, C., Rasmussen, C.: Gaussian processes for regression. In: Advances in Neural Information Processing Systems 8 (NIPS-1995). (1996) 514–520
33. Smola, A., Schölkopf, B., Müller, K.R.: The connection between regularization operators and support vector kernels. Neural Networks **11** (1998) 637–649
34. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley (1994)
35. Bertsekas, D., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific, Belmont, MA (1996)
36. Sutton, R., Barto, A.: Reinforcement Learning, an introduction. MIT Press (1998)
37. Bagnell, D., Kakade, S., Ng, A., Schneider, J.: Policy search by dynamic programming. In: Advances in Neural Information Processing Systems 16 (NIPS-2003). (2004) 831–838
38. Lagoudakis, M., Parr, R.: Reinforcement learning as classification: leveraging modern classifiers. In: Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003). (2003) 424–431
39. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research **6** (2005) 503–556
40. Langford, J., Zadrozny, B.: Relating reinforcement learning performance to classification performance. In: Proceedings of the Twenty-Second International Conference on Machine Learning (ICML-2005). (2005) 473–480
41. Fern, A., Yoon, S., Givan, R.: Approximate policy iteration with a policy language bias: solving relational Markov Decision Processes. Journal of Artificial Intelligence Research **25** (2006) 85–118