Evaluation of Scenario-Generation Methods for Stochastic Programming

Michal Kaut and Stein W. Wallace

in: Pacific Journal of Optimization. See also BibTeX entry below.

BibTeX:

```
@article{KautWall03:2,
  author = {Michal Kaut and Stein W. Wallace},
  title = {Evaluation of Scenario-Generation Methods for Stochastic Programming},
  journal = {Pacific Journal of Optimization},
  year = {2007},
  volume = {3},
  pages = {257--271},
  number = {2},
  classmath = {90C15, 0C31}
}
```


Document created on October 5, 2012.
Cover page generated with CoverPage.sty.

Evaluation of scenario generation methods for stochastic programming

Michal Kaut
michal.kaut@himolde.no *

Stein W. Wallace
stein.w.wallace@himolde.no *

First version: December 2003; this version March 2007

Abstract

Stochastic programs can only be solved with discrete distributions of limited cardinality. Input, however, normally comes in the form of continuous distributions or large data sets. Creating a limited discrete distribution from input is called scenario generation. In this paper, we discuss how to evaluate the quality or suitability of scenario generation methods for a given stochastic programming model. We formulate minimal requirements that should be imposed on a scenario generation method before it can be used for solving the stochastic programming model. We also show how the requirements can be tested.

The procedures for testing a scenario generation method is illustrated on a case from portfolio management.

Keywords: stochastic programming, scenario tree, scenario generation, stability.

AMS subject classifications: 90C15, 0C31.

1 Introduction

In recent years, stochastic programming has gained an increasing popularity within the mathematical programming community. Advances in hardware and software have allowed users to add stochasticity to models that were difficult to solve as deterministic models only a few years ago. In this context, a stochastic programming model can be viewed as a mathematical programming model with uncertainty about the values of some of the parameters. Instead of single values, these parameters are then described by distributions (in a single-period case), or by stochastic processes (in a multi-period case). A single-period stochastic programming model can thus be formulated as:

$$\begin{gathered} z^* = \min_{\boldsymbol{x}\in\boldsymbol{X}} F(\boldsymbol{x}) = \min_{\boldsymbol{x}\in\boldsymbol{X}} \mathbb{E}^G\big[f(\boldsymbol{x},\tilde{\boldsymbol{\xi}})\big] = \min_{\boldsymbol{x}\in\boldsymbol{X}} \int_{\tilde{\boldsymbol{\xi}}} f(\boldsymbol{x},\tilde{\boldsymbol{\xi}})\,\mathrm{d}G(\tilde{\boldsymbol{\xi}})\,, \\ \boldsymbol{x}^* \in \operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{X}} F(\boldsymbol{x})\,,\ \text{so } z^* = F(\boldsymbol{x}^*)\,, \end{gathered} \tag{1}$$

where $\tilde{\boldsymbol{\xi}}$ is a random vector, whose distribution G must be independent of the decision vector $\boldsymbol{x}$. Note that we assume that the feasible region $\boldsymbol{X}$ is independent of $\tilde{\boldsymbol{\xi}}$, i.e. we assume relatively complete recourse.

*Molde University College, P.O. Box 2110, NO-6402 Molde, Norway

Except for some special cases (like the one-period Markowitz mean-variance model), (1) cannot be solved directly with continuous distributions—most solution methods need discrete distributions. In addition, the cardinality of the support of the discrete distribution is limited by the available computing power, as well as the complexity of the decision model. Hence, in most practical applications, the distributions of the stochastic parameters have to be approximated by discrete distributions with a limited number of outcomes. The discretization is usually called a *scenario tree* or an event tree—see Figure 1 for an example. This discretization should in our view be seen as part of the modeling process: it represents our way of modeling the data. Different scenario generation methods model the relationships among the data in different ways, and hence end up with different trees.

As a result, we solve only an approximation of (1), with the quality of the approximation directly linked to the quality of the scenario tree: *garbage in, garbage out* holds here as well. Surprisingly, there has been little focus on measuring the quality of scenario trees. In this paper, we thus ask the question of what is a good scenario generation method for a given stochastic programming model. The link to the decision model is very important, we do not believe there is a scenario generation method that would be best for all possible models, even if these models were subject to the same random phenomena.

When comparing scenario generation methods, we focus on practical performance, not on the theoretical properties: it may be comforting to know that a certain method approximates the distribution perfectly when the number of outcomes goes to infinity, yet it does not mean that the method is good for generating a tree with just a few scenarios. Indeed, some of the common scenario generation methods do not guarantee convergence to the true distribution, but perform very well in real-life problems. For more information on the theoretical properties, see for example [5].

Because of the variety of both scenario generation methods and decision models, we do not provide a guideline of the type "for this model use that method". Instead, we formulate two important properties that a scenario generation method should satisfy in order to be usable for a given model. We also show how to test the properties. The user can thus test several scenario generation methods, and choose the one that is best suited for the given decision model.

An alternative to scenario generation, as discussed in this paper, is obviously what has become known as bounding. Bounding is well outlined in textbooks such as [2, 10]. In those cases the idea is to generate a scenario tree that results in an optimal objective function bounding the true value from above or below. The opposite bound is then found for the given solution. If the bounds are too loose, the tree is refined, and the process is repeated. Approximations, as discussed in this paper, and bounding are alternative methods, and one is not uniformly better than the other.

There is an important qualitative difference between two-stage and multi-stage problems: in a two-stage model, the only "real" decision is taken in the root of the tree, as the second-stage decisions can be obtained by solving the recourse problem. This means that a solution obtained on one scenario tree is feasible for all scenario trees (for the given problem), a property we are going to need for our proposed tests. For most of the paper, we will therefore consider only two-stage (one-period) problems, and come back to the multi-period case in Section 5. For the lack of a better word, we will still call the set of scenarios a "scenario tree", even if it is only a "bush" in the two-stage case.

One of the common notations for stochastic variables is to use a tilde to distinguish between a variable and its value. In our case, however, the situation gets complicated by the fact that a scenario tree is a sample from the "true" distribution, but at the same time we need to speak about the distribution of the values in the tree. We thus introduce the following notation: the "true" random variables are denoted by tilde (as in $\tilde{\xi}$), random variables connected to the scenario tree by a "hat" ($\hat{\xi}$), and single realizations (sample points, scenarios) by letters with no decorations (ξ).

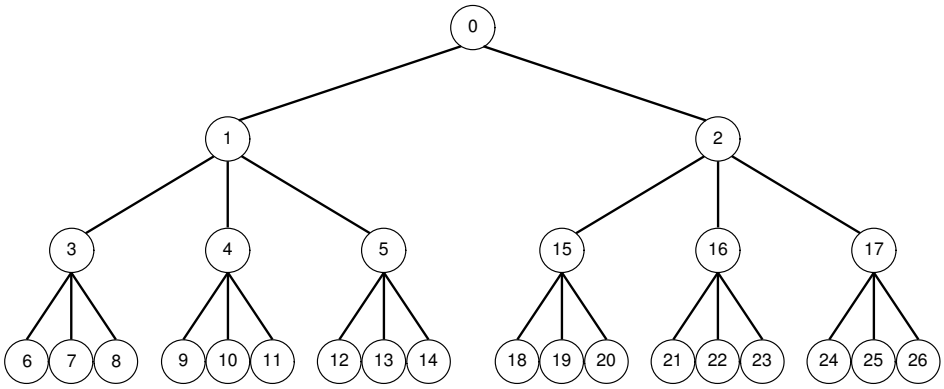

Figure 1: Example of a three-period scenario tree

Following the same logic, the hat is also being used to distinuish between the true and the scenario-based problem, so $\hat{G}$, $\hat{F}$ and $\hat{x}^*$ stand for the scenario-based distribution function, objective function and the optimal solution of the scenario-based problem, respectively. In addition, boldface is used to denote vectors. The notation can combine, so $\hat{\boldsymbol{\xi}}$ denotes a stochastic vector. The scenario-based version of the optimization problem (1) is therefore

$$\begin{aligned} \hat{z}^* = \min_{\boldsymbol{x}\in\boldsymbol{X}} \hat{F}(\boldsymbol{x}) = \min_{\boldsymbol{x}\in\boldsymbol{X}} \mathbb{E}^{\hat{G}}\big[f(\boldsymbol{x},\hat{\boldsymbol{\xi}})\big] = \min_{\boldsymbol{x}\in\boldsymbol{X}} \int_{\hat{\boldsymbol{\xi}}} f(\boldsymbol{x},\hat{\boldsymbol{\xi}})\,\mathrm{d}\hat{G}(\hat{\boldsymbol{\xi}})\,, \\ \hat{\boldsymbol{x}}^* \in \operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{X}} \hat{F}(\boldsymbol{x})\,,\ \text{so } \hat{z}^* = \hat{F}(\hat{\boldsymbol{x}}^*)\,, \end{aligned} \tag{2}$$

The rest of the paper is organized as follows: Section 2 provides two criteria for the quality of a scenario tree, and Section 3 shows how to test them. Section 4 then demonstrates the tests on a case from portfolio management. In Section 5 we discuss the multi-period case, before we finally, in Section 6, discuss some more aspects of scenario generation and conclude the paper.

2 Measures of quality of a scenario tree

We should always remember that our goal is to solve a stochastic program. The only reason we need a scenario tree is that we do not know how to solve the problem directly with the random vector $\tilde{\boldsymbol{\xi}}$. Hence, we should judge a scenario tree (and, consequently, a scenario generation method) by the quality of the decision it gives us. *We are not concerned about how well the distribution is approximated, as long as the scenario tree leads to a "good" decision.* In other words, we are not necessarily searching for a discretization of a distribution that is optimal (or even good) in the statistical sense. See [18] for discussion and examples of this topic.

The error of approximating a random vector $\tilde{\boldsymbol{\xi}}$ with distribution G by a discretization $\hat{\boldsymbol{\xi}}$ with distribution $\hat{G}$, for a given stochastic programming problem (1), is thus given by the *optimality gap*, i.e. the difference between the value of the true objective function at the optimal solutions of the true and the approximated problems:

$$\begin{aligned} e(F,\hat{F}) = e_f(G,\hat{G}) &= F\big(\operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{X}} \hat{F}(\boldsymbol{x})\big) - F\big(\operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{X}} F(\boldsymbol{x})\big) \\ &= F\big(\operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{X}} \hat{F}(\boldsymbol{x})\big) - \min_{\boldsymbol{x}} F(\boldsymbol{x}) = F(\hat{\boldsymbol{x}}^*) - z^*\,, \end{aligned} \tag{3}$$

where $\hat{F}(\boldsymbol{x}) = \mathbb{E}^{\hat{G}}\left[f(x,\tilde{\boldsymbol{\xi}})\right]$ and $\hat{\boldsymbol{x}}^*$ is the optimal solution of the approximated problem. If there are more optimal solutions to either of the problems, both $\hat{\boldsymbol{x}}^*$ and the argmin functions should be understood as one of the optimal solutions. The subscript f in $e_f(G,\hat{G})$ is used to stress the dependence on the optimization problem.

Note that $e_f(G,\hat{G}) \geq 0$, since the second term is the true minimum, while the first is the value of the (true) objective function at an approximate solution. Note also that we do not compare the optimal solutions, but the corresponding values of the objective function. The reason is that the objective function of a stochastic programming problem typically is flat, so there can be different solutions giving very similar objective values. We do not think it wise to declare solutions leading to (almost) the same objective function values as representing instability.

Definition (3) has one rather obvious problem: the optimality gap is, in most practical problems, impossible to calculate. In [17] Pflug solves this by proving that, under certain uniform Lipschitz conditions,

$$e_f(G,\hat{G}) \leq 2 \sup_{\boldsymbol{x}\in\boldsymbol{X}} \left|\hat{F}(\boldsymbol{x}) - F(\boldsymbol{x})\right| \leq 2Ld(\hat{G},G)\,,$$

where L is a Lipschitz constant of f and $d(\hat{G},G)$ is a Wasserstein (transportation) distance of the distribution functions $\hat{G}$ and G. An algorithm is then developed to construct a scenario tree that minimizes the upper bound, i.e. the Wasserstein distance $d(\hat{G},G)$.

It is worth noting that these bounds can, in general, be quite loose, so even if we find a scenario tree that minimizes the upper bound, there is no guarantee that we will be close to the minimum of e_f. The reason is that minimization of the upper bound does not depend on the optimization problem, so we have lost the link between the scenario generation and the problem. (Only the constant L, i.e. the tightness of the bound, depends on the problem.)

In this paper, we have therefore taken a different approach: Instead of trying to find the optimal scenario generation method, we focus on the evaluation of a given method. This allows users who already have one or more scenario generation methods implemented to test how they perform for a given optimization problem. In this context, a scenario generation method may be seen as a heuristic for minimizing the optimality gap e_f. Of course, also methods that set out to minimize an upper bound on e_f may be tested. Hence, our approach does not imply any initial views on what are good methods: rather, we try to find a neutral way of evaluating any suggested method.

3 Testing a scenario generation method

Ideally, we would like to estimate the optimality gap $e_f(G,\hat{G})$ and therefore test the requirement that the scenario-based solution is not significantly suboptimal, i.e.

$$e_f(G,\hat{G}) \approx 0\,.$$

The problem is that testing this property is in most practical problems impossible, since it needs solving the optimization problem with the (true) random vector—and if we could solve that, we would not need scenario trees in the first place.

In some cases, however, it is possible to perform some approximate tests. One possibility is to built a *reference tree*, and use it as a representation (approximation) of the true random vector. Typically, such a tree should be as big as possible, i.e. the biggest tree for which we can still solve the optimization problem. To create such a tree, we would need a method that is guaranteed to be unbiased—we cannot use the method we want to test! For example, if we use a data series as input for the scenario generation, we may try using the history as scenarios. Or, if it is possible, we could

sample as big a tree as we are able to solve the problem with—the solution time hardly matters as the problem with a reference tree has to be solved only once.

When we do not have any reference tree and thus cannot compute the true optimal objective value $z^* = \min_{\boldsymbol{x} \in \boldsymbol{X}} F(\boldsymbol{x})$, we may still be able to *compare* two scenario generation methods, as long as we can *evaluate* the true objective function F: the method that gives the lowest true value is the better one for a given problem.

3.1 Stability requirements

Having a small optimality gap $e_f(G, \hat{G})$ is the major goal of scenario generation. However, if we are unable to test this property, there are other weaker requirements that can and should be present in a scenario generation method. Since most of the common methods are stochastic and generate different scenario trees on every run, the first requirement is stability: if we generate several trees (with the same input) and solve the optimization problem with these trees, we should get (approximately) the same optimal objective function value. This may also be seen as a robustness requirement on the scenario generation method.

It may seem that the group of scenario generation methods involving randomness is rather limited, including mostly methods based on sampling. Our experience suggests, however, that the group is, in fact, rather large. As an example consider the moment-matching methods: for a given number of random variables and a given number of moments, there is a minimal number of scenarios s_{min} needed to match all the moments. This number is usually quite low, so we will use $s > s_{min}$ (or even $s \gg s_{min}$) scenarios for a better approximation—see [7] for illustration of the negative effects of moment matching with only s_{min} scenarios. Consequently, there are infinitely many trees matching the input specifications, and some form of randomness can be expected in choosing between them. For example, the methods presented in [8, 9] both include some randomness and can thus be subject to the presented tests.

There are, however, scenario generation methods that do not include any randomness, for example the "optimal-discretization" method from [7, 17]. For these methods, the stability requirement does not seem to be an issue at all. Yet, even with these methods there is one source of instability: the number of scenarios. Since the number of scenarios is usually arbitrary, minor differences should not affect the solution of the stochastic programming model significantly. We can thus do the same tests as with the stochastic methods, as long as we are able to generate several trees with different numbers of scenarios.

Out-of-sample stability

This requirement can be stated as follows: if we generate several scenario trees (discretizations $\hat{\boldsymbol{\xi}}$) for a given random vector $\tilde{\boldsymbol{\xi}}$, and solve the stochastic programming problem with each tree, we should get (approximately) the same value of the true objective function. We use the term "out-of-sample" because the stability is judged on a different sample than was used for finding the solution.

Let us say that we generate K scenario trees $\hat{\boldsymbol{\xi}}_k$, solve the optimization problem with each one of them, and obtain optimal solutions $\hat{\boldsymbol{x}}_k^*$, $k = 1 \ldots K$. The stability requirement can then be written in the following (equivalent) ways:

$$
\begin{aligned}
F\left(\hat{\boldsymbol{x}}_k^*\right) &\approx F\left(\hat{\boldsymbol{x}}_l^*\right) \\
F\left(\operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{X}} \hat{F}_k\left(\boldsymbol{x}\right)\right) &\approx F\left(\operatorname*{argmin}_{\boldsymbol{x}\in\boldsymbol{X}} \hat{F}_l\left(\boldsymbol{x}\right)\right) \\
e_f(G,\hat{G}_k) &\approx e_f(G,\hat{G}_l)
\end{aligned}
$$

where $k,l \in \{1,\ldots,K\},\ k \neq l$.[1] From the last equation it is immediately obvious that if the optimality gap $e_f(G,\hat{G})$ is negligible then the method is out-of-sample stable, as

$$
e_f(G,\hat{G}_k) \approx 0 \quad \forall k \in \{1,\ldots,K\}.
$$

The problem with this definition of stability is that we can test it only if we are able to evaluate the true objective function $F\left(\boldsymbol{x}\right)$. If this is not the case, we can use another, weaker, form of out-of-sample stability: if we have two scenario trees $\hat{\boldsymbol{\xi}}_k$ and $\hat{\boldsymbol{\xi}}_l$ with distributions $\hat{G}_k$ and $\hat{G}_l$, $k \neq l$, and their respective optimal solutions $\hat{\boldsymbol{x}}_k^*$ and $\hat{\boldsymbol{x}}_l^*$, then a stable method implies

$$
\begin{aligned}
\hat{F}_k\left(\hat{\boldsymbol{x}}_k^*\right) &\approx \hat{F}_k\left(\hat{\boldsymbol{x}}_l^*\right), \\
\hat{F}_l\left(\hat{\boldsymbol{x}}_l^*\right) &\approx \hat{F}_l\left(\hat{\boldsymbol{x}}_k^*\right),
\end{aligned}
$$

and finally

$$
\hat{F}_k\left(\hat{\boldsymbol{x}}_l^*\right) \approx \hat{F}_l\left(\hat{\boldsymbol{x}}_k^*\right).
$$

Note that these are still an out-of-sample-type tests, since we test the solutions on different samples/trees than were used to find them.

In-sample stability

This requirement guarantees that whichever scenario tree we choose, the optimal value of the objective function reported by the model itself is (approximately) the same:

$$
\hat{F}_k\left(\hat{\boldsymbol{x}}_k^*\right) \approx \hat{F}_l\left(\hat{\boldsymbol{x}}_l^*\right)
$$

or, equivalently,

$$
\min_{\boldsymbol{x}\in\boldsymbol{X}} \hat{F}_k\left(\boldsymbol{x}\right) \approx \min_{\boldsymbol{x}\in\boldsymbol{X}} \hat{F}_l\left(\boldsymbol{x}\right),
$$

for all $k,l \in \{1,\ldots,K\},\ k \neq l$. This is called "in-sample" because the solutions are evaluated on the sample (tree) they came from. Since this stability depends only on the optimization model, it can—and therefore should—always be checked.

It should be noted that in-sample stability does not have a direct connection to our target of $e_f(G,\hat{G}) \approx 0$, in the sense that it is possible to have $e_f(G,\hat{G}) = 0$ while being in-sample unstable: this can happen when all the scenario solutions $\hat{\boldsymbol{x}}_k^*$ are equal to the true solution $\boldsymbol{x}^*$, but the scenario-tree objective functions $\hat{F}_k$ are different, so $\hat{F}_k\left(\hat{\boldsymbol{x}}_k^*\right) = \hat{F}_k\left(\boldsymbol{x}^*\right)$ is different for each $k \in \{1,\ldots,K\}$.

It is important to realize that the two stabilities are different and that there is no simple relationship between them. This can be demonstrated on the following one-period, one-dimensional example:

[1] Again, if there are more optimal solutions to any of the optimization problems, the argmin should be understood as taking one of them, typically the one we get from the solver.

$$\min_{x\in\mathbb{R}} F(x) = \mathbb{E}^G\left[\left(x-\tilde{\xi}\right)^2\right]$$

This problem can be solved explicitly, for any distribution of $\tilde{\xi}$ (we drop the distribution superscript):

$$F(x) = \mathbb{E}\left[\left(\tilde{\xi}-x\right)^2\right] = \mathrm{Var}\left[\tilde{\xi}\right] + \left(x-\mathbb{E}\left[\tilde{\xi}\right]\right)^2,$$

i.e. "mean square error = variance + bias squared". The optimal solution is therefore

$$x^* \in \operatorname*{argmin}_{x\in\mathbb{R}} F(x) = \mathbb{E}\left[\tilde{\xi}\right]$$
$$F(x^*) = \min_{x\in\mathbb{R}} F(x) = \mathrm{Var}\left[\tilde{\xi}\right]$$

Now, assume we generate sample trees $\hat{\xi}_k,\ k = 1\dots K$, and get solutions $\hat{x}_k^* = \mathbb{E}\left[\hat{\xi}_k\right]$. Let us first assume that the scenario generation method is such that all the samples $\hat{\xi}_k$ have the correct means (i.e. $\mathbb{E}\left[\hat{\xi}_k\right] = \mathbb{E}\left[\tilde{\xi}\right]$), but the variances are different in all the samples. Hence $\hat{F}_k\left(\hat{x}_k^*\right) = \mathrm{Var}\left[\hat{\xi}_k\right]$ is different for all the samples, so we do not have in-sample stability. At the same time, $\hat{x}_k^* = x^*$, so $F\left(\hat{x}_k^*\right) = F(x^*)$, and the out-of-sample stability holds.

If we instead assume that we have a scenario generation method that produces samples with correct variances (i.e. $\mathrm{Var}\left[\hat{\xi}_k\right] = \mathrm{Var}\left[\tilde{\xi}\right]$), but the means are different in all the samples,[2] we would have $\hat{F}_k\left(\hat{x}_k^*\right) = \mathrm{Var}\left[\hat{\xi}_k\right] = \mathrm{Var}\left[\tilde{\xi}\right]$, so we would have the in-sample stability. On the other hand, $F\left(\hat{x}_k^*\right) = \mathrm{Var}\left[\tilde{\xi}\right] + \left(\mathbb{E}\left[\hat{\xi}_k\right] - \mathbb{E}\left[\tilde{\xi}\right]\right)^2$ would be different for all the samples, so the problem would be out-of-sample unstable.

We may ask what is the practical difference between the in-sample and out-of-sample stability, and which of them is more important to have. Having out-of-sample stability means that the real performance of the solution $\hat{\boldsymbol{x}}_k^*$ is stable, i.e. it does not depend on which scenario tree $\boldsymbol{\xi}_k$ we choose. However, if we do not have in-sample stability as well, we may be getting good solutions without knowing how good they really are (unless we do the out-of-sample evaluation). The opposite (in-sample without out-of-sample stability) is even more dangerous, since the real performance of the solutions depends on which scenario tree we pick—without the possibility of detecting it by solving the problem on several trees. Another important difference is that the in-sample tests are easily applicable on multi-period problems as well, since we only compare the reported optimal objective values. To do any kind of out-of-sample tests is, on the other hand, much more difficult in the multi-period case, as we will see in Section 5.

In the example above, we saw that it is possible to have in-sample instability in the objective function, but still have in-sample stability of the solutions—in our case, the solutions were the same in all the sample trees. This obviously guarantees an out-of-sample stability. Therefore, if we detect an in-sample instability of the objective, we should look at the solutions as well. However, it does not work the other way around, i.e. we can have the out-of-sample stability even if the in-sample solutions vary, because the objective functions of stochastic programming problems are typically flat.

To conclude the section, we would like to repeat that stability is the minimal requirement we should put on a scenario generation method. Hence, before we start to work with a new optimization model, or a new scenario generation method (remember that we test the two together), we should always run the stability tests: the in-sample test and, if feasible, some out-of-sample tests.

[2] This may not be a very realistic example, but that is not the point here.

3.2 Estimating the optimality gap

In addition to the stability requirements presented in this paper, it is also possible—at least to some degree—to estimate the optimality gap. For example, [14] presents a stochastic estimator of the optimality gap and an algorithm to compute its confidence intervals. In this section we summarize the relevant results from the paper, omitting all the details and assumptions.

First of all, the authors show that if we have a scenario tree $\hat{\boldsymbol{\xi}}$ which was sampled from the true distribution $\tilde{\boldsymbol{\xi}}$ and solve the scenario-based problem (2), then

$$\mathbb{E}[\hat{z}^*] \leq z^*, \tag{4}$$

where the expected value is over possible scenario trees $\hat{\boldsymbol{\xi}}$. This means that the scenario-based problem overestimates its own performance. Moreover, they show that the error decreases monotonically with increasing size of the scenario tree.

Since sampling yields unbiased distributions, we also get

$$F(\boldsymbol{x}) = \mathbb{E}\left[\hat{F}(\boldsymbol{x})\right], \tag{5}$$

for any feasible solution $\boldsymbol{x}$, where the expected value is again taken over scenario trees. This allows us to estimate the optimality gap for *any feasible solution* $\boldsymbol{x}$ in the following way:

First, we *sample* N scenario trees $\hat{\boldsymbol{\xi}}_k$, $k = 1 \ldots N$ and solve the associated optimization problems, storing the optimal (in-sample) objective values $\hat{z}_k^*$. In addition, we have to compute the value of the scenario-based objective functions $\hat{F}_k$ for the given solution $\boldsymbol{x}$, i.e. $\hat{F}_k(\boldsymbol{x})$. From (4) and (5) it then follows that

$$F(\boldsymbol{x}) = \mathbb{E}\left[\hat{F}(\boldsymbol{x})\right] \approx \frac{1}{N}\sum_{k=1}^{N} \hat{F}_k(\boldsymbol{x})$$

$$z^* \geq \mathbb{E}\left[\min_{\boldsymbol{x}' \in \boldsymbol{X}} \hat{F}_k(\boldsymbol{x}')\right] \approx \frac{1}{N}\sum_{k=1}^{N} \hat{z}_k^*,$$

which finally gives us the estimate of the optimality gap for $\boldsymbol{x}$ as

$$F(\boldsymbol{x}) - z^* \leq \mathbb{E}\left[\hat{F}(\boldsymbol{x}) - \min_{\boldsymbol{x}' \in \boldsymbol{X}} \hat{F}_k(\boldsymbol{x}')\right] \approx \frac{1}{N}\sum_{k=1}^{N} \left[\hat{F}_k(\boldsymbol{x}) - \hat{z}_k^*\right]. \tag{6}$$

The optimality gap $e_f(G, \hat{G})$ for a scenario tree $\hat{\boldsymbol{\xi}}$ with distribution function $\hat{G}$ is then given by the optimality gap for the associated optimal solution $\hat{\boldsymbol{x}}^*$ of (2), i.e. by replacing $\boldsymbol{x}$ by $\hat{\boldsymbol{x}}^*$ in (6). Note that this is only a stochastic upper bound of the optimality gap, so the actual computed value can even be negative. It is, however, possible to use the result to compute the confidence interval for the optimality gap, see [14] for details.

A similar method, together with computational results, can be found in [13]. In addition, [1] presents an updated version of the presented method, which only needs one or two replications of the sampled scenario tree, instead of N needed in the original method.

3.3 Improving the performance

When the testing shows that our scenario generation method is unstable or that the optimality gap is too big (for the given stochastic programming model), the next question is what are the possible causes of the problems? The answer depends to a large degree on the type of the scenario generation method used:

Sampling methods.

When we use a sampling method, the strongest candidate for the source of the problem is a lack of scenarios, as we know that, with an increasing number of scenarios, the discrete distribution converges to the true distribution. Hence, by increasing the number of scenarios, the trees will be closer to the true distribution, and consequently also closer to each other. As a result, both the instability and the optimality gap should decrease.

In addition to increasing the number of scenarios (which is usually limited by the solution time for the optimization model), we can also try to improve the sampling method—see for example [12, 16].

Moment-matching methods.

With moment-matching methods, the situation is more complicated. Since these methods generally do not guarantee convergence, increasing the number of scenarios is not guaranteed to help. We thus need to look at different issues. In the following discussion we assume that in all the tested scenario trees $\hat{\xi}_k$, we have managed to match all the required properties perfectly, i.e. the source(s) of the problem must be among the properties we do not control (and that can, therefore, vary among the tested trees).

Even without the convergence guarantee, the first test is still the number of scenarios: There is an obvious difference between a discrete distribution with three points, and a discrete distribution with a thousand points, even if their first four (or even more) moments can be equal. How important the difference is depends very much on the particular implementation of the moment-matching procedure. For optimization-based methods like [8], the gain from extra scenarios seems to be quite limited. On the other hand, the transformation-based method from [9] benefits from the increased number of scenarios, because it uses the extra freedom to increase the *smoothness* of the distribution. The smoothness has value since it, most likely, represents many statistical properties that we do not enforce explicitly. The transformations do not preserve these properties explicitly, nor do they totally ruin them.

The important issue of the moment-matching methods is whether we match the right properties—an issue that obviously depends on the optimization model. While for a mean-variance model it is enough to match the means, variances, and the correlation matrix, most optimization models will require more. Our experience shows that the first four moments are often a good enough description of the margins, at least for financial models. On the other hand, a correlation matrix may not be enough to describe the multi-variate structure. In such a case, we may try to match also higher co-moments, or use a copula ([4, 15]), if we have the necessary data and a scenario generation method that can work with these properties.

What shall we then do when we discover that a moment-matching method is either instable or leads to strongly suboptimal solutions? The first attempt should be to increase the number of scenarios as much as possible (we still have to be able to solve the optimization model in a reasonable time). If this helps, the problems were probably caused by the lack of smoothness in the original trees. Otherwise, it means that there is some property the decision model reacts to, but we do not control it in the scenario generation process. We have no general advice on identifying the missing property—it depends on the decision model, and is typically done by trial-and-error, based on problem understanding.

4 Test case: a portfolio optimization

As a test case, we use a simple one-period portfolio optimization problem: we consider one-month investments in three indices (stocks, short-term bonds, and long-term bonds), in four markets (USA, UK, Germany, Japan), giving us twelve assets in total. We model the situation of a US investor, so we have to include the exchange rates of the three foreign currencies to USD. Hence, we have fifteen random variables in the scenario trees. In the model, we do not allow short positions. In addition, it is possible to hedge the currency risk with forward contracts.

As an objective function, we use the expected return and quadratic penalties for shortfalls (returns under a given threshold):

$$\begin{aligned} \mathrm{sf}(\boldsymbol{\xi}) &= \max\bigl(\mathrm{Tg} - \mathrm{ret}(\boldsymbol{x}, \boldsymbol{\xi}), 0\bigr) \\ F(\boldsymbol{x}) &= \mathbb{E}\left[\mathrm{ret}(\boldsymbol{x}, \tilde{\boldsymbol{\xi}}) - \alpha\bigl(\mathrm{sf}(\tilde{\boldsymbol{\xi}}) + \beta\,\mathrm{sf}(\tilde{\boldsymbol{\xi}})^2\bigr)\right], \end{aligned}$$

where α is a risk-aversion parameter, and β is a weight of the quadratic term. In the test, we used the following values: $\mathrm{Tg} = 0$, $\alpha = 1$, and $\beta = 10$.

We use the moment-matching scenario generation method from [9] to generate the scenarios. This method generates scenario trees with specified first four moments of the marginal distributions (mean, standard deviation, skewness and kurtosis), and correlation matrix.

For the test, we use a large one-period scenario tree as our representation of the "real world". Based on this "benchmark scenario set", we compute the moments and correlations of the differentials $\frac{\xi - \xi_0}{\xi_0}$ of the data for the margins (univariate components) $\tilde{\xi}$ of the random vector $\tilde{\boldsymbol{\xi}}$, where $\boldsymbol{\xi}_0$ is the vector of values at the root of the tree. The moments and correlations then constitute the targets to match with our scenario generation procedure.

The benchmark scenario tree is also used for the out-of-sample testing. *It is important that the benchmark set is provided exogenously, that is, it is not generated by the same method which we want to test.* In our case, the benchmark scenario set (tree) was generated by a method based on principal component analysis described in [19]. The benchmark tree has 20,000 scenarios, and is based on data in the period from January 1990 to April 2001. See [11] for a detailed description of the properties of the benchmark scenario set.

Since we have the benchmark scenario set as a representation of the true distribution, we can perform all the tests from Section 3: For a given size of the tree, we generate 25 scenario trees, solve the optimization model on each of them, and then evaluate the solutions on the benchmark tree. This is repeated for several different sizes of scenario trees.

Results of the test are presented in Table 1. We see that the scenario generation method used gives a reasonable stability, both in-sample and out-of-sample. We see also that the out-of-sample values have a smaller variance than the in-sample values. The reason is that in-sample tests evaluate (different) solutions on different trees, while the out-of-sample tests evaluate all the solutions on the common benchmark tree. Note also that the performance (true objective value of the solutions) improves as the number of scenarios increases.

Another important observation is that in the case of 50 scenarios, the in-sample objective values are significantly higher than the out-of-sample (true) values. In other words, the solution is notably worse than the model tells us. This is a common observation: when we do not have enough scenarios, the model overestimates the quality of its own solution. Note that this is in concordance with (4), even if the inequality is actually provable only for unbiased methods such as sampling. Only out-of-sample evaluations can tell us how good a solution really is.

Table 1: Stability tests for the optimization model. For every size of the scenario tree, 25 trees were generated, and the model was solved on each of them. The solutions were then evaluated on the benchmark tree to obtain the out-of-sample values. The table presents sample means and standard deviations of the optimal values, for the different sizes.

description of the test			# of scenarios			
type of test	objective f.	value	50	250	1000	5000
in-sample	$\hat{F}_k\left(\hat{\boldsymbol{x}}_k^*\right)$	average	0.009 48	0.009 36	0.009 31	0.009 29
		std. dev.	0.000 23	0.000 11	0.000 05	0.000 02
out-of-sample	$F\left(\hat{\boldsymbol{x}}_k^*\right)$	average	0.009 02	0.009 26	0.009 28	0.009 30
		std. dev.	0.000 15	0.000 03	0.000 01	0.000 00

In addition to the stability tests, we have solved the optimization model on the benchmark tree, and obtained the "true" optimal value: 0.009 30. Hence, we see that the optimality gap is nonexistent in the case of 5000 scenarios, quite small with 250 scenarios, but rather noticeable with 50 scenarios.

The conclusion of the test case is that the tested scenario generation method is suitable for the given optimization model: it is stable and has a small optimality gap, provided we have enough scenarios.

5 Multi-period trees

So far, we have only discussed two-stage (one-period) problems. In this section, we discuss the applicability of the tests for multi-stage problems. We focus only on the out-of-sample tests, since the in-sample tests can be done in the same way as for two-stage programs.

Before we start, however, we need to introduce some more notation. Since we have more than one period, the stochastic vectors $\tilde{\boldsymbol{\xi}}$ now become discrete-time stochastic processes $\{\tilde{\boldsymbol{\xi}}_t\}_{t\in\boldsymbol{T}}$, where $\boldsymbol{T}$ is the set of stages. Note that since choosing the stages is a natural part of the modeling process, we assume that the time discretization has already been done and we thus have the set $\boldsymbol{T}$. To simplify the notation, we will drop the subscript $t \in \boldsymbol{T}$. In addition, we assume that $\boldsymbol{T} = \{0,\ldots,T\}$.

A scenario tree is then a sample from $\{\tilde{\boldsymbol{\xi}}_t\}$ and is denoted simply by $\{\hat{\boldsymbol{\xi}}_t\}$. To be able to traverse the tree, however, we need additional notation: we denote by n_t the number of nodes in the tree $\{\hat{\boldsymbol{\xi}}_t\}$ at stage t, with $t=0,\ldots,T$ and $n_0=1$. In every stage, the nodes are indexed from one, so (n,t) is the n-th node at stage t, $n \le n_t$. We use a superscript for the node index, so $\boldsymbol{\xi}_t^n$ denotes the value of $\{\hat{\boldsymbol{\xi}}_t\}$ at the node (n,t), and $\boldsymbol{x}_t^n$ is the vector of optimal decisions at this node (only for $t<T$). Since $n_0=1$, we omit the node index for the root and write $\boldsymbol{\xi}_0$ and $\boldsymbol{x}_0$.

As pointed out earlier, the problem with the multi-period case is that we can no longer evaluate the true objective function $F(\boldsymbol{x})$ for a given x: the vector of solutions $\boldsymbol{x}_t^n$ corresponds to the tree $\{\hat{\boldsymbol{\xi}}_t\}$ and does not coincide with $\{\tilde{\boldsymbol{\xi}}_t\}$ for $t>0$. This problem did not exist in the one-period case where we had decision variables only in the root—which is the same for all the trees.

One way of getting the solution $\boldsymbol{x}_t^n$ is therefore to solve the model on a tree with $\boldsymbol{\xi}_t^n$ as the root. The "true" objective function $F(\boldsymbol{x})$ is then evaluated in the following way:

- We start with a discrete representation $\{\zeta_t^n,\ n=1\ldots N_t,\ t=0\ldots T\}$ of the true stochastic process $\{\tilde{\boldsymbol{\xi}}_t\}$. This may be either the supplied reference tree, or a sample from the simulator. In any case, we should have $N_t \gg n_t$.

- For every node ζ_t^n of the reference, generate a scenario tree $\{\hat{\xi}_t\}$ with the root $\xi_0 = \zeta_t^n$.
- Solve the stochastic programming model on this scenario tree and store the root solution x_0 as a solution corresponding to node ζ_t^n.
- Evaluate the objective value on $\{\zeta_t^n\}$, using the stored values x_t^n.

How the scenario trees $\{\hat{\xi}_t\}$ are generated depends on our representation of the true stochastic process $\{\tilde{\xi}_t\}$: if we have a simulator of the stochastic process $\{\tilde{\xi}_t\}$, we can generate a whole T-period tree for every node ζ_t^n. Note that in this case, the trees for $t > 0$ are also likely to depend on the "parent" nodes $\zeta_{t'}^n$, $t' < t$. This approach is known as the "rolling horizon", see for example [6] for more information.

If we do not have a simulator for $\{\tilde{\xi}_t\}$ and only have a reference tree $\{\zeta_t^n\}$, we are no longer able to generate T-period trees with roots $\xi_0 = \zeta_0$, since we have no information about the distribution for $t > T$. Hence, we can only generate $(T-t)$-period trees, based on the distribution of the subtree of the node ζ_t^n. This can be done only in the case when the model can be solved on subtrees of the original scenario tree. This means that the decisions in every node of the tree may only depend on the predecessors of the node—excluding thus models with constraints going through scenarios, such as constraints on the (overall) expected return.

Once we are able to evaluate the true objective function $F(x)$, we can perform the stability tests described in Section 3.1.

If we do not have any representation of the true stochastic process $\{\tilde{\xi}_t\}$, or cannot evaluate the true objective function $F(x)$ for some other reason, we can still perform a test based on two scenario trees $\{\hat{\xi}_{kt}\}$, $\{\hat{\xi}_{lt}\}$, $k \neq l$. This test is analogous to the one described in Section 3.1. Just as before, we start by finding the optimal solutions $\hat{x}_k^*$, $\hat{x}_l^*$, based on the trees $\{\hat{\xi}_{kt}\}$, $\{\hat{\xi}_{lt}\}$. Unlike the one-period case, we cannot just plug x_l^* into the tree $\{\hat{\xi}_{kt}\}$, because the trees do not coincide beyond the root. We can, however, fix x_0 to the value of the root solution $\hat{x}_{l,0}^*$ and solve the optimization problem on the tree $\{\hat{\xi}_{kt}\}$. If we denote the resulting solution $\hat{x}_k^* | x_0 = \hat{x}_{l,0}^*$, a stable method implies

$$\hat{F}_k(\hat{x}_k^*) \approx \hat{F}_k\left(\hat{x}_k^* | x_0 = \hat{x}_{l,0}^*\right).$$

If we, in addition, repeat the same procedure for the solution $\hat{x}_k^*$ and the tree $\{\hat{\xi}_{lt}\}$, we get another stability requirement

$$\hat{F}_l\left(\hat{x}_l^* | x_0 = \hat{x}_{k,0}^*\right) \approx \hat{F}_k\left(\hat{x}_k^* | x_0 = \hat{x}_{l,0}^*\right).$$

In this way, we only test stability of the root solution x_0. This is not, however, such a big problem, as the root solution is usually the only one that gets implemented. At $t = 1$, we typically update the scenario tree, resolve the model, and implement the new root solution. However, if it is desirable to evaluate the whole solution vector (also called a policy), we refer the reader to [3].

6 How far can we get?

So far, we have implicitly assumed that all distributions are known. In reality this is very rarely the case. What does this lack of knowledge mean, particularly for the issue we raise here, that of generating good scenario trees? First, let us distinguish between three cases:

- The distributions are fully known.

- We have theoretical knowledge about the distribution family, plus data.
- We only have data.

Although it is common to assume in theoretical papers that a distribution is fully known—very often this is done indirectly by basing the paper (or the tests within the paper) on certain distributions—this is in our view not the case in many applications. An exception may be planning under well-known stochasticity, such as the roll of a (fair) die or the flip of a (fair) coin. But most interesting applications concern real-life phenomena such as price, demand or quality. So, if the distribution is not known, what can we then say about scenario trees? The most important, and also obvious, observation is that certain theoretical properties of scenario generation methods become less useful. For example, sample-based methods will, if the sample is large enough, produce scenario trees arbitrarily close to the distribution from which we sample. But how useful is it to know that we have convergence towards something that is not the real thing?

This becomes even more important if we do not know the distribution family. A common approach in this case is to assume some family. If we do so, we *know* that sampling will converge to a distribution that contains information we have added without foundation in data or theory. An alternative is using the empirical distribution directly. In this way we avoid adding any subjective information to the data. On the other hand, this approach will prevent the use of any methodology that requires knowledge of the distribution itself.

In the (scarce) occasions that we have distributional information, it is normally advisable to estimate the distribution, since otherwise that information is lost. But we still face the problem of having convergence to a distribution that may not be the right one.

But there is more to the problem than this. To use data we have to assume that the past is a reasonable description of the future. Whether this is the case cannot be tested, it is a question of belief. We can of course test, at least in many cases, whether or not the past would have been a good description of the future, in the past. But it is still a question of belief if this is still the case.

We have mentioned simulation as a way to evaluate the true value of a certain decision. The good aspect of simulation models is that they can be allowed to contain details that we are unable to put into the optimization model. But all the problems discussed above remain. Within the simulation model we need to sample from distributions, and they are normally not fully known.

This discussion may seem to be very negative, we seem to be saying that nothing works. That is not the point. The point is to be aware of the shortcomings of modeling in general, and stochastic programming in particular. We can get to a certain point, but thereafter empirical testing becomes impossible, and we have to start believing in what we do. And in our view, this also means that we should be very skeptical of high accuracy statements from models. We may know that a given method retains two digits of accuracy, but we cannot know how many correct digits were in the input.

Hence, the convergence properties are not so important in real-life applications. Instead, it is more important to have scenario generation tools that give us reasonable control over the tree with a limited number of scenarios. What we want is a method that is stable, has a small optimality gap and produces small trees. But there is a limit to what we require in terms of accuracy, given these properties.

A small warning may be appropriate: The conclusion of this paper is not that moment-matching is a good scenario generation method for every stochastic program—we know that is not true. Our test case shows only that (our implementation of) moment-matching scenario generation works well with our optimization model, at least when we can afford to have enough scenarios. We do not say that there is not a better method for our model, nor that the method works as well for other problems. We have simply shown that the given scenario generation procedure, with the given data, produces

scenario trees passing the stability tests for the given optimization problem. In our view such tests should always be performed.

Conclusions

In this paper, we have discussed how to evaluate the suitability of a given scenario generation method for a given stochastic programming problem. We have identified the main properties the scenario generation method should satisfy, and suggested ways to test them. We have also demonstrated the methodology on a test case from portfolio management.

References

[1] Güzin Bayraksan and David P. Morton. Assessing solution quality in stochastic programs. *Mathematical Programming*, 108(2–3):207–634, sep 2006.

[2] John R. Birge and François Louveaux. *Introduction to stochastic programming*. Springer-Verlag, New York, 1997. ISBN 0-387-98217-5.

[3] Anukal Chiralaksanakul and David P. Morton. Assessing policy quality in multi-stage stochastic programs. Stochastic Programming E-Print Series, `http://www.speps.info`, 2004.

[4] Robert T. Clemen and Terence Reilly. Correlations and copulas for decision and risk analysis. *Manag. Sci.*, 45(2):208–224, February 1999.

[5] Jitka Dupačová, Giorgio Consigli, and Stein W. Wallace. Scenarios for multistage stochastic programs. *Ann. Oper. Res.*, 100:25–53, 2000. ISSN 0254-5330.

[6] Stein-Erik Fleten, Kjetil Høyland, and Stein W. Wallace. The performance of stochastic dynamic and fixed mix portfolio models. *European J. Oper. Res.*, 140(1):37–49, 2002. ISSN 0377-2217.

[7] Ronald Hochreiter and Georg Ch. Pflug. Financial scenario generation for stochastic multi-stage decision processes as facility location problems. *Ann. Oper. Res.*, 152(1):257–272, 2007.

[8] K. Høyland and S. W. Wallace. Generating scenario trees for multistage decision problems. *Manag. Sci.*, 47(2):295–307, 2001.

[9] Kjetil Høyland, Michal Kaut, and Stein W. Wallace. A heuristic for moment-matching scenario generation. *Comput. Optim. Appl.*, 24(2-3):169–185, 2003. ISSN 0926-6003.

[10] P. Kall and S.W. Wallace. *Stochastic Programming*. Wiley, Chichester, 1994.

[11] Michal Kaut, Stein W. Wallace, Hercules Vladimirou, and Stavros Zenios. Stability analysis of a portfolio management model based on the conditional value-at-risk measure. Feb 2003.

[12] R. R. P. Kouwenberg. Scenario generation and stochastic programming models for asset liability management. *European J. Oper. Res.*, 134(2):51–64, 2001.

[13] Jeff T. Linderoth, Alexander Shapiro, and Stephen J. Wright. The empirical behavior of sampling methods for stochastic programming. *Ann. Oper. Res.*, 142(1):215–241, 2006.

[14] W.K. Mak, D.P. Morton, and R.K. Wood. Monte carlo bounding techniques for determining solution quality in stochastic programs. *Oper. Res. Lett.*, 24:47–56, 1999.

[15] Roger B. Nelsen. *An Introduction to Copulas*. Springer-Verlag, New York, 1998.

[16] T. Pennanen and M. Koivu. Integration quadratures in discretization of stochastic programs. Stochastic Programming E-Print Series, `http://www.speps.info`, May 2002.

[17] G. C. Pflug. Scenario tree generation for multiperiod financial optimization by optimal discretization. *Math. Program.*, 89(2):251–271, 2001.

[18] James E. Smith. Moment methods for decision analysis. *Manag. Sci.*, 39(3):340–358, March 1993.

[19] N. Topaloglou, Vladimirou H., and S. A. Zenios. CVaR models with selective hedging for international asset allocation. *J. Banking Finance*, 26(7):1535–1561, 2002.