Scenario Generation for Stochastic Programming

A Practical Introduction

Michal Kaut
michal.kaut@himolde.no

Molde University College

Stochastics in Logistics and Transportation,
Håholmen, June 10–12, 2006

Outline

Outline

Where Do Scenarios Come From?

A stochastic programming (SP) problem is a math. programming problem, with values of some parameters replaced by distributions.
Hence, to solve the problem, we need:

- A model describing the problem.
- Values of the deterministic (known) parameters.
- Description of the stochasticity.
    - Known distributions, described by densities and/or CDFs.
    - Historical data, i.e. a discrete sample.
    - Only some properties of the distributions, for ex. moments.

Problem: SP can handle only discrete samples of limited size, so we need to approximate the distribution. The approximation is called a scenario tree.

Where Do Scenarios Come From?

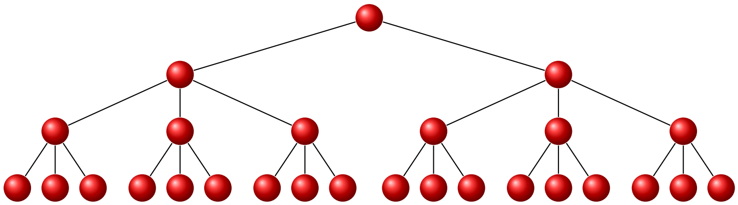
A stochastic programming (SP) problem is a math. programming problem, with values of some parameters replaced by distributions.

Hence, to solve the problem, we need:

- A model describing the problem.
- Values of the deterministic (known) parameters.
- Description of the stochasticity.
  - Known distributions, described by densities and/or CDFs.
  - Historical data, i.e. a discrete sample.
  - Only some properties of the distributions, for ex. moments.

Problem: SP can handle only discrete samples of limited size, so we need to approximate the distribution. The approximation is called a scenario tree.

Scenario Tree – Example and Terminology.



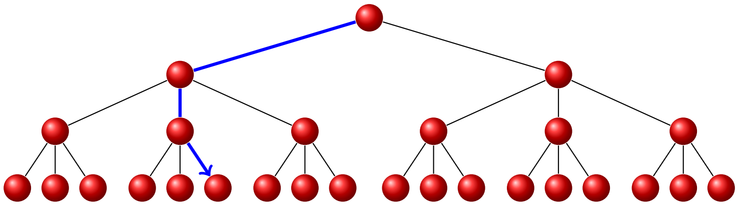
Terminology:

scenario is a path from the root to one leaf.

stage is a moment in time, when decisions are taken.

period is a time interval between two stages.

Scenario Tree – Example and Terminology.



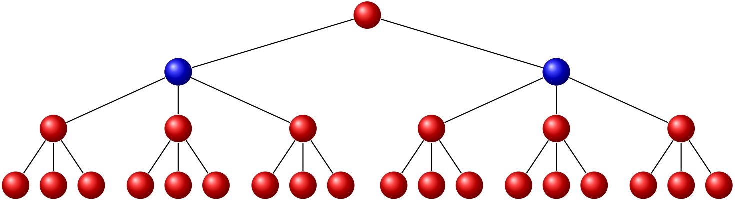
Terminology:

scenario is a path from the root to one leaf.

stage is a moment in time, when decisions are taken.

period is a time interval between two stages.

Scenario Tree – Example and Terminology.



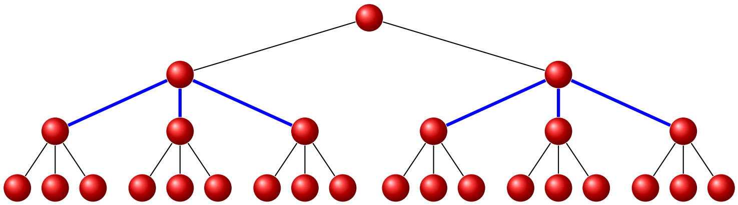
Terminology:

scenario is a path from the root to one leaf.

stage is a moment in time, when decisions are taken.

period is a time interval between two stages.

Scenario Tree – Example and Terminology.



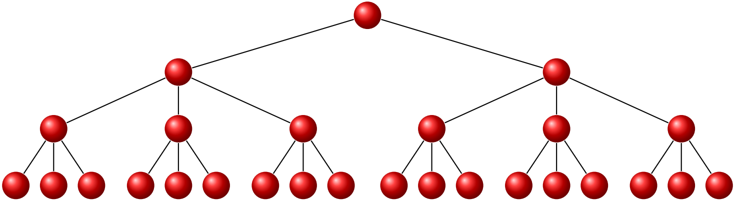
Terminology:

scenario is a path from the root to one leaf.

stage is a moment in time, when decisions are taken.

period is a time interval between two stages.

Scenario Tree – Example and Terminology.



Terminology:

scenario is a path from the root to one leaf.

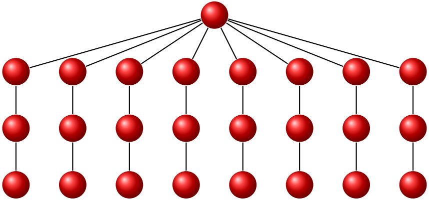
stage is a moment in time, when decisions are taken.

period is a time interval between two stages.

Tree above: $2 \times 3 \times 3 = 18$ scenarios, 4 stages, and 3 periods.

Scenario Tree – Importance of Branching.

Why a tree, why not a “fan” like this?



- Branching = arrival of new information.
- Fan above: no new information after the first stage.
- Hence, the fan represents a two-stage problem.

What to Do Before Scenario Generation

Prior to scenario generation, we have to:

- Decide the time discretization.
  - number of stages
  - lengths of time periods
- Know what information becomes available when, relative to the timing of decisions.
  This issue does not exist in the deterministic case.
- Decide the number of branches per stage.
  - Some methods will do this automatically.

Note: deep vs. wide trees – an open question. . .

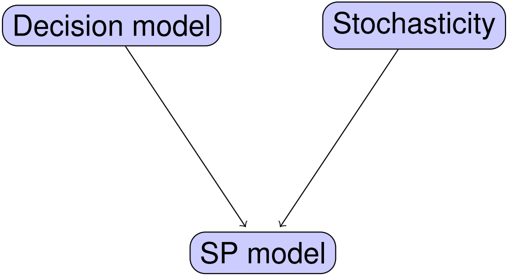
Sources of Data for Scenarios

- Historical data
  - Is history a good description of the future?
- Simulation based on a mathematical/statistical model
  - Parameters estimated from the real case
- Expert opinion
  - Subjective
  - Back-testing is not possible.

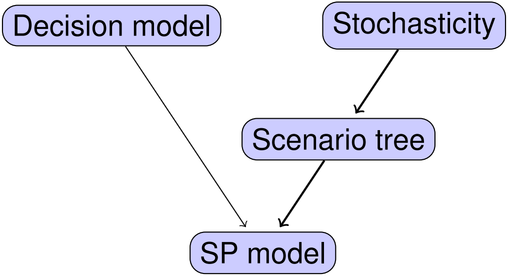
- Often a *combination* of more of the above
  - Estimate the distribution from historical data,
    then use a mathematical model and/or an expert opinion
    to adjust the distribution to the current situation.

Outline

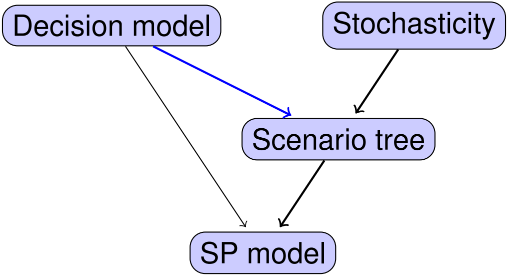
Structure of a Stochastic Programming Problem



Structure of a Stochastic Programming Problem



Structure of a Stochastic Programming Problem



- Choice of a good scenario generation method is problem-dependent.
- Scenario generation is a part of the modelling process.

Specifics of Scenario Generation

- Scenarios are not a natural part of the problem, but a result of the method.
- Hence, neither the user nor the modeler are typically interested in scenarios and their generation.
  - Until recently, many users of stochastic programming did not pay much attention to scenario generation.
- BUT: Scenarios can influence the quality of the solution (garbage in – garbage out).
- One solution is to make scenario generation as much automatic, i.e. “invisible” to the user, as possible.

A Good Scenario Generation Method Should

- Be as automatic (hidden to the user) as possible.
- Influence the solution only as little as possible.
- The scenario-based solution should converge to the true optima, with increasing number of scenarios.
- Be as “good” as possible for a given number of scenarios.

- The distance from the true distribution in the statistical sense is not so important.
- What is important is problem-dependent: for example, for the classical one-period Markowitz mean-variance model, it is enough to capture means, variances, and covariances, the rest is irrelevant.

A Good Scenario Generation Method Should

- Be as automatic (hidden to the user) as possible.
- Influence the solution only as little as possible.
- The scenario-based solution should converge to the true optima, with increasing number of scenarios.
- Be as “good” as possible for a given number of scenarios.

- The distance from the true distribution in the statistical sense is not so important.
- What is important is problem-dependent: for example, for the classical one-period Markowitz mean-variance model, it is enough to capture means, variances, and covariances, the rest is irrelevant.

Outline

Quality of Scenario Trees and How to Measure It

In accessing the quality, we have consider two things:

Error

- We use an approximation of the true distribution, so we are likely to find a suboptimal solution.
- Not straightforward how to measure the error.

Stability

- If we generate several scenario trees, the solutions should not vary *too much*.
- Stochastic programs tend to have *flat objective functions*, so we can only require stability of the objective values, not of the solutions themselves.

Some Notation

The original (unsolvable) problem

$$\min_{\boldsymbol{x} \in \boldsymbol{X}} F\left(\boldsymbol{x}; \tilde{\xi}\right)$$

is replaced by a scenario-based problem

$$\min_{\boldsymbol{x} \in \boldsymbol{X}} F\left(\boldsymbol{x}; \tilde{\boldsymbol{\eta}}\right).$$

In the stability tests, we generate several scenario trees $\tilde{\boldsymbol{\eta}}_k, \; k = 1, \ldots, n$, leading to solutions

$$x_k^* = \operatorname*{argmin}_{\boldsymbol{x} \in \boldsymbol{X}} F\left(\boldsymbol{x}; \tilde{\boldsymbol{\eta}}_k\right).$$

Error Caused by the Discretization

Pflug (2001) defines an approximation error caused by $\tilde{\eta}_k$ as:

$$\begin{aligned} \mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) &= F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\eta}_k); \tilde{\xi}\Big) - F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi}); \tilde{\xi}\Big) \\ &= F\Big(\boldsymbol{x}_k^*; \tilde{\xi}\Big) - \min_{\boldsymbol{x}} F\big(\boldsymbol{x}; \tilde{\xi}\big) \geq 0\,. \end{aligned}$$

To evaluate $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k)$, we would need to:

- Evaluate the "true" objective function $F(\boldsymbol{x}; \tilde{\xi})$.
  - Can sometimes be done using a "simulator".
- Solve the original problem, i.e. $(\arg)\min_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi})$.
  - Impossible …Otherwise, we would not need scenarios.

Error Caused by the Discretization

Pflug (2001) defines an approximation error caused by $\tilde{\eta}_k$ as:

$$\begin{aligned} \mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) &= F\left(\operatorname*{argmin}_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\eta}_k); \tilde{\xi}\right) - F\left(\operatorname*{argmin}_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi}); \tilde{\xi}\right) \\ &= F\left(\boldsymbol{x}_k^*; \tilde{\xi}\right) - \min_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi}) \geq 0\,. \end{aligned}$$

To evaluate $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k)$, we would need to:

- Evaluate the "true" objective function $F(\boldsymbol{x}; \tilde{\xi})$.
  - Can sometimes be done using a "simulator".
- Solve the original problem, i.e. $(\arg)\min_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi})$.
  - Impossible . . . Otherwise, we would not need scenarios.

Error Caused by the Discretization

Pflug (2001) defines an approximation error caused by $\tilde{\eta}_k$ as:

$$\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) = F\left(\operatorname*{argmin}_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\eta}_k); \tilde{\xi}\right) - F\left(\operatorname*{argmin}_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi}); \tilde{\xi}\right)$$
$$= F\left(\boldsymbol{x}_k^*; \tilde{\xi}\right) - \min_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi}) \geq 0\,.$$

To evaluate $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k)$, we would need to:

- Evaluate the "true" objective function $F(\boldsymbol{x}; \tilde{\xi})$.
  - Can sometimes be done using a "simulator".
- Solve the original problem, i.e. $(\mathrm{arg}) \min_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi})$.
  - Impossible ... Otherwise, we would not need scenarios.

Outline

Tests Using a Simulator

Assume that we have a "simulator" for evaluating $F(\boldsymbol{x}; \tilde{\xi})$,
i.e. the true performance of a solution $\boldsymbol{x}$.
This allows us to:

- Compare two solutions $\boldsymbol{x}_1^*$, $\boldsymbol{x}_2^*$.
- Compare two different scenario-generation methods.
- Test an out-of-sample stability of a given method:
  1. Generate a set of trees $\tilde{\eta}_k$, $k = 1, \ldots, n$.
  2. Solve problems using the trees $\rightarrow$ solutions $\boldsymbol{x}_k^*$.
  3. Test whether $F(\boldsymbol{x}_k^*; \tilde{\xi}) \approx F(\boldsymbol{x}_l^*; \tilde{\xi})$
  - The test is equivalent to $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) \approx \mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_l)$.
  - Without stability, we have a problem!

Tests Using a Simulator

Assume that we have a "simulator" for evaluating $F(\boldsymbol{x}; \tilde{\xi})$,
i.e. the true performance of a solution $\boldsymbol{x}$.
This allows us to:

- Compare two solutions $\boldsymbol{x}_1^*$, $\boldsymbol{x}_2^*$.
- Compare two different scenario-generation methods.
- Test an out-of-sample stability of a given method:
  1. Generate a set of trees $\tilde{\eta}_k,\ k = 1, \ldots, n$.
  2. Solve problems using the trees $\rightarrow$ solutions $\boldsymbol{x}_k^*$.
  3. Test whether $F(\boldsymbol{x}_k^*; \tilde{\xi}) \approx F(\boldsymbol{x}_l^*; \tilde{\xi})$

  - The test is equivalent to $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) \approx \mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_l)$.
  - Without stability, we have a problem!

Notes on the Stability Test

- $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) \approx 0$ implies $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) \approx \mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_l)$ and stability.
- Stability test assumes that we get a different tree on each run of the scenario-generation method.
- Otherwise, we can run it with different tree sizes.

Another issue:

- Only the root variables can be moved from one tree to another, as the scenarios do not coincide.
- To evaluate $F\left(\boldsymbol{x}; \tilde{\xi}\right)$, we have to fix the root part of $\boldsymbol{x}$ and (re)solve the problem.
- Not such a big issue, as the root variables (decisions) are the only ones implemented.

Notes on the Stability Test

- $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) \approx 0$ implies $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) \approx \mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_l)$ and stability.
- Stability test assumes that we get a different tree on each run of the scenario-generation method.
- Otherwise, we can run it with different tree sizes.

Another issue:

- Only the root variables can be moved from one tree to another, as the scenarios do not coincide.
- To evaluate $F\left(\boldsymbol{x}; \tilde{\xi}\right)$, we have to fix the root part of $\boldsymbol{x}$ and (re)solve the problem.
- Not such a big issue, as the root variables (decisions) are the only ones implemented.

Out-of-Sample Tests Without a Simulator

Instead of using a simulator, we can "cross test", i.e. test

$$F\left(\boldsymbol{x}_k^*; \tilde{\boldsymbol{\eta}}_l\right) \;\text{ for }\; l \neq k$$

for all $k = 1, \ldots, n$.

- It is still an out-of-sample test, as we test the solutions on different trees than were used to find them.
- If we have to choose one of the solutions $\boldsymbol{x}_k$, we would choose the most stable one.

In-Sample Stability

Instead of the true performance, we look at the optimal objective value reported by the problem itself:

$$F\left(\boldsymbol{x}_k^*; \tilde{\boldsymbol{\eta}}_k\right) \approx F\left(\boldsymbol{x}_l^*; \tilde{\boldsymbol{\eta}}_l\right),$$

or, equivalently,

$$\min_{\boldsymbol{x}} F\left(\boldsymbol{x}; \tilde{\boldsymbol{\eta}}_k\right) \approx \min_{\boldsymbol{x}} F\left(\boldsymbol{x}; \tilde{\boldsymbol{\eta}}_l\right).$$

- No direct connection to out-of-sample stability.
  - Can even have $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}) = 0$, without in-sample stability.
- Without this, we can not trust the reported performance of the scenario-based solutions.

What If We Do Not Have Stability?

What does it mean:

- No stability $\rightarrow$ decision depends on the choice of the tree.

What to do:

- Change/improve the scenario generation method.
- Increase the number of scenarios.
- Generate several trees, get the solutions and then "somehow" choose the best solution.

Example

Note: A proper theoretical treatment of stability can be found in Heitsch, Römisch and Strugarek (2006).

Outline

One-Period Case - Standard Sampling I.

Univariate random variable

- This is a standard random number generation.
- Methods exist for all possible distributions.

Independent multivariate random vector

- Generate one margin at a time, combine all against all
  - guaranteed independence
  - grows exponentially with the dimension
  - trees need often some “pruning” to be usable
- Generate one margin at a time, then join together, first with first, second with second. . .
  - independent only in the limit
  - size independent on the dimension

One-Period Case - Standard Sampling II.

General multivariate case

- Special methods for some distributions.
  - Ex.: normal distribution via Cholesky decomposition
- Use principal components to get “independent” variables.
  - Components are independent only for normal variables.
  - Generally, they are only uncorrelated.

Bootstrapping / Sampling from historical data

- Does not need any distributional assumptions.
- Needs historical data.
- Are historical data a good description of the future?

One-Period Case - Standard Sampling II.

General multivariate case

- Special methods for some distributions.
  - Ex.: normal distribution via Cholesky decomposition
- Use principal components to get "independent" variables.
  - Components are independent only for normal variables.
  - Generally, they are only uncorrelated.

Bootstrapping / Sampling from historical data

- Does not need any distributional assumptions.
- Needs historical data.
- Are historical data a good description of the future?

Handling Multiple Periods

Generate one single-period subtree at a time.
Start in the root, move to its children, and so on.

Inter-temporal independence

- Easy, as the distributions does not change.

Distribution depends on the history.

- Distribution of children of a node depends on the values on the path from the root to that node.
- The dependence is modeled using stochastic processes like ARMA, GARCH, ...
- Effects we might want to consider/model:
  - mean reversion
  - variance increase after a big jump

Sampling Methods – Summary

Pros

- Easy to implement.
- Distribution converges to the true one.

Cons

- Bad performance/stability for small trees.
  - This can be improved by using corrections or some special techniques, such as *low-discrepancy sequences*.
- Have to know the distribution to sample from.

Outline

Property-Matching Methods – Basic Info

- These methods construct the scenario trees in such a way that a given set of properties is matched.
- The properties are for ex. moments of the marginal distributions and covariances/correlations.
- Typically, the properties do not specify the distributions fully; the rest is left to the method.
  - Different methods produce very different results.
  - The issue is very significant for bigger trees, with many more degrees of freedom.

Example 1 – from Høyland and Wallace (2001)

- An optimization problem with values of the random variables and scenario probabilities as variables.
- The measured properties are expressed as function of these variables.
- The objective is to minimize a distance (usually L_2) of these properties from their target values.
- Leads to highly non-linear, non-convex problems.

Example

- Works well for small trees, otherwise very slow.
- The optimization is often underspecified & no control what the solver does about the extra degrees of freedom.

Example 2 – from Høyland, Kaut, Wallace (2003)

- Developed as a fast approximation to the previous method, in the case of four marginal moments + correlations.
- Build around two transformations:
  1. Correcting correlations
     - Multiply the random vector by a Cholesky component
     - Changes also the marginal distributions (except normal)
  2. Correcting the marginal distributions
     - A cubic transformation of the margins, one margin at a time
     - Changes the correlation matrix
- The two transformations are repeated alternately.
- Starting point can be, for ex., a correlated normal vector.

- Works well for large trees (creates *smooth* distributions).
- Needs pre-specified probabilities (usually equiprobable).

Details

Property-Matching Methods – Summary

Pros

- Do not have to know/assume a distribution family, only to estimate values of the required properties.
- Can combine historical data with today's predictions.
- The marginal distributions can have very different shapes, so the vector does not follow any standard distribution.

Cons

- No convergence to the true distribution.
- If we know the distribution, we can not utilize this information, i.e. we throw it away.
- Can be hard to find which properties to use.

Outline

"Optimal Discretization" by G. Pflug I.

Starts with the approximation error $\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k)$:

$$\begin{aligned} \mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) &= F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\eta}_k); \tilde{\xi}\Big) - F\Big(\operatorname*{argmin}_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi}); \tilde{\xi}\Big) \\ &= F\Big(\boldsymbol{x}^*_k; \tilde{\xi}\Big) - \min_{\boldsymbol{x}} F(\boldsymbol{x}; \tilde{\xi}) \geq 0 \,. \end{aligned}$$

Pflug (2001) shows that, under certain Lipschitz conditions,

$$\mathrm{e}_f(\tilde{\xi}, \tilde{\eta}_k) \leq 2 \sup_{\boldsymbol{x}} \Big| F(\boldsymbol{x}; \tilde{\eta}_k) - F(\boldsymbol{x}; \tilde{\xi}) \Big| \leq 2\, L\, \mathrm{d}(\tilde{\eta}_k, \tilde{\xi}) \,,$$

where L is a Lipschitz constant of $f()$, with $F(\boldsymbol{x}; \tilde{\xi}) = \mathbb{E}^{\tilde{\xi}}\Big[f(\boldsymbol{x}, \tilde{\xi})\Big]$ and $\mathrm{d}(\tilde{\eta}_k, \tilde{\xi})$ is a Wasserstein (transportation) distance of distribution functions of $\tilde{\eta}_k$ and $\tilde{\xi}$.

"Optimal Discretization" by G. Pflug II.

The method then creates a scenario tree that minimizes the transportation distance $\mathrm{d}(\tilde{\eta}_k, \tilde{\xi})$.

- Whole multi-period tree is generated at once.
- The tree is "optimal" in a clearly specified sense.
- The optimisation problem disappeared from the measure!
- Nothing is said about tightness of the bounds.

Outline

Step-Wise Growing & Cutting Methods

- Methods to transform a given scenario tree into a tree better suited for the given problem.
- Can have different starting points:
  - A big sampled tree – then we need to reduce it.
  - One scenario only – then we need to grow it.
  - A “fan” (collection of paths) – need to make a tree out of it.
- Differ in the level of integration with the model.
  - Some work only with the distribution of the tree.
  - Some use the model to evaluate the quality.
  - Some are actually a part of the solution method.

These methods are useful if we have a data
(or a simulation model that produces the “data”)
in a form that is *not suitable for the optimization model*.

Scenario Reduction by Dupačová et al

- Starts with discrete probability measure P (a scenario tree) often sampled from stochastic processes (time series).
- The goal is to find a discrete probability measure Q (a smaller tree) of *given cardinality*, that is closest to P in the sense of *Fortet-Mourier* type metric.
  - The metric is independent on the optimization problem.
- The evaluation of the distance leads to a linear transportation problem.
- Need heuristics for deciding which scenarios to remove from the scenario tree given by P. Details
- The results of their numerical example were:
  - 50% scenarios give 90% relative accuracy.
  - 2% scenarios give 50% relative accuracy.

Making Tree Out of a Fan by Heitsch and Römisch

- In practice, data are often available as sequences/paths.

  Examples: rainfall data – each year is a scenario/path.
  : evolving a stochastic process.
- Put together, they form a fan, not a tree.
- Need to bind some nodes into, one to create a *tree*.
- Based on the same ideas as the scenario reduction, i.e. minimizes the *Fortet-Mourier* type metric.
- Two different method, backward and forward. Details
- The results of their numerical example were:
  - 15% nodes give 60% accuracy.
  - 6% nodes give 50% accuracy.

Other methods are also available: clustering / "bucketing".

Model-Based Cutting and Growing Methods

Iterative methods, with a general structure:

loop
  improve the tree: cut and/or grow branches
  solve the problem on the new tree
  analyze the solution
while solution/tree not good enough

- Can start with a single scenario, for ex. expected values.
- New scenarios added by (importance) sampling.
- Example: EVPI-based method by Dempster and Thomson.
  - EVPI = Expected value of perfect information.
  - Uses EVPI to decide where to add/delete scenarios.

Internal Sampling Methods

- Sampling of scenarios is a part of the solution procedure.
- The information where to add/delete scenarios is obtained from the model, for example from the *dual variables*.
  - Then it works only for linear stochastic programs!
- Scenario generation disappears from the modelling process, yet we still have to decide the number of periods etc.

Examples:

Stochastic decomposition by Higle and Sen.

Importance sampling within Benders by Dantzig and Infanger.

Stochastic quasi-gradient method by Ermoliev, Gaivoronski.

- This works for convex programs, not only LPs.

Outline

Summary

- Scenario generation is an important, even if often overlooked, part of stochastic programming.
- A bad scenario-generation method can spoil the result of the whole optimization.
- There is an increasing choice of methods, but one has to test which one works best for a given problem.

- Open questions:
  - Is there a universally good scenario-generation method?
  - What is the optimal structure of a tree (deep vs. wide)?

For Further Reading I

- George B. Dantzig and Gerd Infanger.
  Large-scale stochastic linear programs—importance sampling and Benders decomposition.
  In *Computational and applied mathematics, I (Dublin, 1991)*, pages 111–120. North-Holland, Amsterdam, 1992.

- M. A. H. Dempster and R. T. Thompson.
  EVPI-based importance sampling solution procedures for multistage stochastic linear programmes on parallel MIMD architectures.
  Annals of Operations Research, 90:161–184, 1999.
  also in Proceedings of the POC96 Conference, Versailles, March, 1996.

- Jitka Dupačová, Nicole Gröwe-Kuska, and Werner Römisch.
  Scenario reduction in stochastic programming.
  Mathematical Programming, 95(3):493–511, 2003.

For Further Reading II

Yury M. Ermoliev and Alexei A. Gaivoronski.
Stochastic quasigradient methods for optimization of discrete event systems.
Ann. Oper. Res., 39(1-4):1–39 (1993), 1992.

H. Heitsch and W. Römisch.
Scenario tree modelling for multistage stochastic programs.
Technical Report Preprint 296, DFG Research Center MATHEON, “Mathematics for key technologies", Technische Universität Berlin, Germany, 2005.

H. Heitsch, W. Römisch, and C. Strugarek.
Stability of multistage stochastic programs.
Technical Report Preprint 324, DFG Research Center Matheon "Mathematics for key technologies", 2006.
Accepted for publication in SIAM Journal in Optimization.

For Further Reading III

J.L. Higle and S. Sen.
Stochastic decomposition: A statistical method for large scale stochastic linear programming.
Kluwer Academic Publishers, Dordrecht, 1996.

K. Høyland and S. W. Wallace.
Generating scenario trees for multistage decision problems.
Management Science, 47(2):295–307, 2001.

Kjetil Høyland, Michal Kaut, and Stein W. Wallace.
A heuristic for moment-matching scenario generation.
Computational Optimization and Applications, 24(2-3):169–185, 2003.

Michal Kaut and Stein W. Wallace.
Evaluation of scenario-generation methods for stochastic programming.
Stochastic Programming E-Print Series, `http://www.speps.info`, May 2003.

For Further Reading IV

G. C. Pflug.
Scenario tree generation for multiperiod financial optimization by optimal discretization.
Mathematical Programming, 89(2):251–271, 2001.

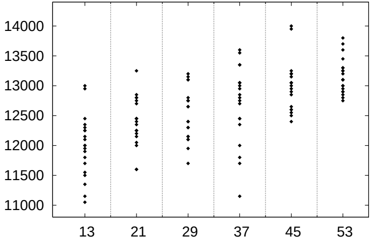
W. Römisch and H. Heitsch.
Scenario reduction algorithms in stochastic programming.
Computational Optimization and Applications, 24(2-3):187–206, 2003.

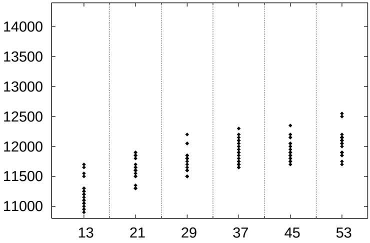
The End

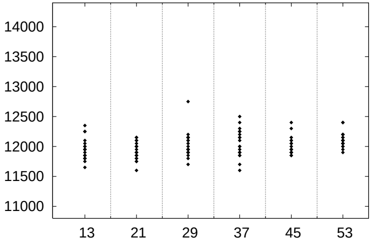
Example: What Is the Best Method and/or Solution?

In-sample stability
of three different methods.

Shows the optimal objective values for different sizes of scenario trees.



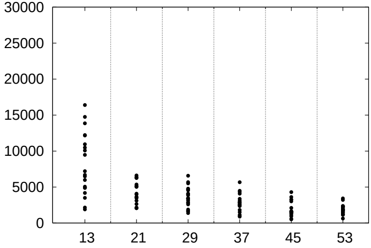


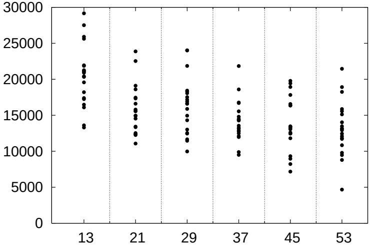


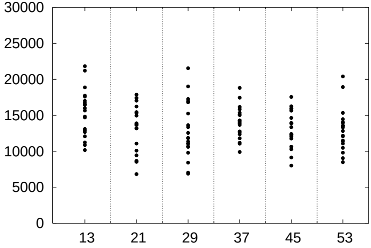
Example: What Is the Best Method and/or Solution?

Out-of-sample
of three different methods.

Shows a level of infeasibility of the solutions for different sizes of scenario trees.

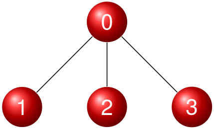






Go Back

Example of the Optimization-Based Moment Matching



$\forall i : (x_i, y_i);\ p_i\,.$

2 variables x, y + node probabilities p

Specifications:

- $\mathbb{E}[x], \mathbb{E}[y];\ \mathbb{E}\left[x^2\right], \mathbb{E}\left[y^2\right];\ \mathrm{Cov}(x,y)$
- Possibly other functions of x, y, p.

$$\begin{aligned}\min_{x,y,p}\ & \Big(\sum_i p_i x_i - \mathbb{E}[x]\Big)^2 + \Big(\sum_i p_i y_i - \mathbb{E}[y]\Big)^2 \\ & + \Big(\sum_i p_i x_i^2 - \mathbb{E}\left[x^2\right]\Big)^2 + \Big(\sum_i p_i y_i^2 - \mathbb{E}\left[y^2\right]\Big)^2 \\ & + \Big(\sum_i p_i (x_i - \mathbb{E}[x])(y_i - \mathbb{E}[y]) - \mathrm{Cov}(x,y)\Big)^2 \\ \text{s.t.:}\ & \sum_i p_i = 1 \quad \text{and} \quad p_i \geq 0,\ i = 1,\dots,3\,.\end{aligned}$$

Go Back

More Info on Transformation-Based Moment Matching

Correction of the correlations

- The target correlation matrix is $R_* = L_* L_*^T$.
- The correlation matrix at step k is $R_k = L_k L_k^T$.
- Then $Y = L_* L_k^{-1} X$ has correlation matrix R_*.

The cubic transformation

- For each margin i: $Y_i = a + bX_i + cX_i^2 + dX_i^3$
- To find the coefficients a, b, c, d, we have to:
  - express the moments of Y_i as a function of a, b, c, d and the moments of X;
  - find the values of a, b, c, d that minimize the L_2 distance of the moments from their target values.
- This is a non-linear, non-convex optimization problem fortunately with only four variables.

Go Back

Scenario-Reduction Heuristics I.

- Methods to remove k of the N scenarios.
- Then have to adjust probabilities of the rest of the tree.

Backward reduction

- Remove $k_1 \leq k$ scenarios, minimizing over all $k_i \leq k$ and over all combinations of scenarios to be removed.
- If $k_1 < k$, repeat with $k_2 \leq k - k_1$, etc.

Backward reduction of single scenarios

- Variant of the above, with $k_i = 1$ (one scenario at a time).

Forward selection

- Choosing the remaining $N - k$ scenarios.
- Selection is done recursively, one at a time.
- This method gets very slow for bigger trees P and Q: about $1000\times$ slower for $N = 3^6 = 729$ and $N - k = 600$.

Scenario-Reduction Heuristics II.

Two additional heuristics were created Heitsch and Römisch, improving on the performance.

Simultaneous backward reduction

- The major difference is to include *all deleted scenarios* into each backward step *simultaneously*.
- Better results (smaller distance from P) than the original backward reduction, but slower.
- Running time *decreases* with the size of Q, for given P.

Fast forward selection

- An improvement of the forward method.
- This methods yields the best trees.
- Running times are comparable to the above, but the running time *increases* with the size of Q.

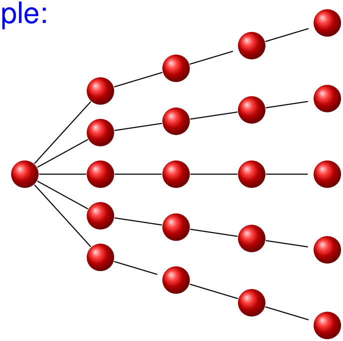
Go Back

Scenario Tree Modelling – Methods + Example

Backward tree construction

- Start at the last stage, join some nodes into one.
- This joins all their predecessors as well.
- Move backwards until the first stage.

Example:

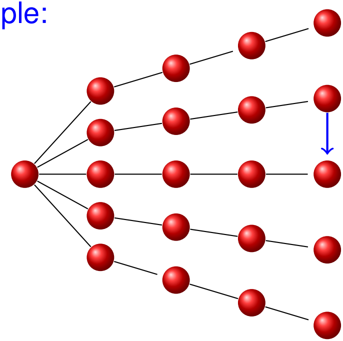


Scenario Tree Modelling – Methods + Example

Backward tree construction

- Start at the last stage, join some nodes into one.
- This joins all their predecessors as well.
- Move backwards until the first stage.

Example:

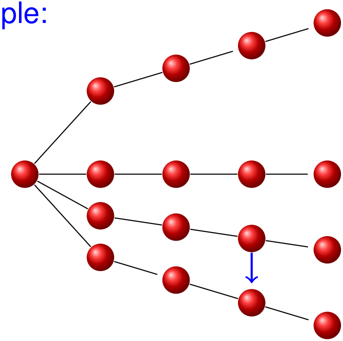


Scenario Tree Modelling – Methods + Example

Backward tree construction

- Start at the last stage, join some nodes into one.
- This joins all their predecessors as well.
- Move backwards until the first stage.

Example:

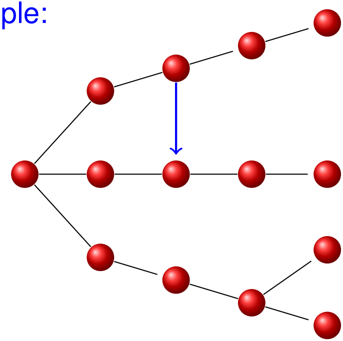


Scenario Tree Modelling – Methods + Example

Backward tree construction

- Start at the last stage, join some nodes into one.
- This joins all their predecessors as well.
- Move backwards until the first stage.

Example:

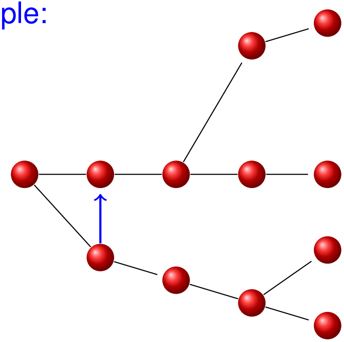


Scenario Tree Modelling – Methods + Example

Backward tree construction

- Start at the last stage, join some nodes into one.
- This joins all their predecessors as well.
- Move backwards until the first stage.

Example:

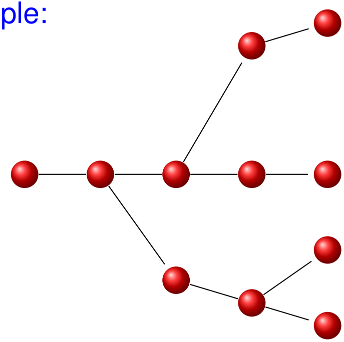


Scenario Tree Modelling – Methods + Example

Backward tree construction

- Start at the last stage, join some nodes into one.
- This joins all their predecessors as well.
- Move backwards until the first stage.

Example:



Scenario Tree Modelling – Methods + Example

Backward tree construction

- Start at the last stage, join some nodes into one.
- This joins all their predecessors as well.
- Move backwards until the first stage.

Forward tree construction

- Start at the first stage, join some nodes into one.
- Move forward until the last stage.

Comparison of the methods:

- Almost no difference in the speed.
- With equal settings, the forward method creates trees with (much) *less nodes and scenarios*.
  (Joining nodes in the last stage means deleting scenarios.)

Go Back