
AIMMS Language Reference - Stochastic Programming

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com or order your hard-copy at www.lulu.com/aimms.

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

Paragon Decision Technology B.V.	Paragon Decision Technology Inc.	Paragon Decision Technology Pte.
Schipholweg 1	500 108th Avenue NE	Ltd.
2034 LS Haarlem	Ste. # 1085	80 Raffles Place
The Netherlands	Bellevue, WA 98004	UOB Plaza 1, Level 36-01
Tel.: +31 23 5511512	USA	Singapore 048624
Fax: +31 23 5511517	Tel.: +1 425 458 4024	Tel.: +65 9640 4182
	Fax: +1 425 458 4025	

Email: info@aimms.com
WWW: www.aimms.com

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. $\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, and $\text{A}_{\text{M}}\text{S}_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.

In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.

This documentation was typeset by Paragon Decision Technology B.V. using $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and the LUCIDA font family.

Chapter 19

Stochastic Programming

The mathematical programming types discussed so far have a common assumption that all the input data used in the formulation of the mathematical program is known with certainty. This is known as “decision making under certainty,” and the corresponding models are called *deterministic* models. Models that account for uncertainty in the input data are called *stochastic* models, and the theory and techniques used to solve stochastic models is commonly referred to as stochastic programming. You can find an introduction to stochastic programming in Chapters 16 and 17 of the AIMMS Optimization Modeling Guide. A more in-depth discussion of stochastic programming and its solution methods can be found, for instance, in [Bi97] and [Ka05].

Deterministic vs. stochastic models

In this chapter, you will find a description of the facilities built into AIMMS for creating and solving stochastic models. From any existing deterministic linear (LP) or mixed-integer (MIP) model, AIMMS is able to automatically create a stochastic model as well, *without the need for you to reformulate any of the constraint definitions*. The only steps necessary to create a stochastic model are

Stochastic programming in AIMMS

- to indicate which parameters and variables in your deterministic model are to become stochastic in a declarative manner, and
- to provide the scenario tree and the stochastic input data.

Being able to generate both a deterministic and stochastic model from an identical symbolic formulation allows for any changes you make in the deterministic formulation to automatically propagate to the stochastic model. This significantly reduces the effort involved with maintaining a stochastic model associated with a given deterministic model.

Single formulation

Section 19.1 discusses a number of basic concepts in stochastic programming. These provide a common understanding necessary for the introduction of the stochastic programming facilities of AIMMS discussed in Section 19.2. Section 19.3 describes the facilities available in AIMMS for the generation of a scenario tree, while Section 19.4 discusses the steps necessary to solve a stochastic model in AIMMS.

This chapter

19.1 Basic concepts

In this section you will find a number of basic concepts that are commonly used in stochastic programming. They will help you to unambiguously understand the stochastic programming facilities in AIMMS.

Basic concepts

In stochastic programming *stages* define a collection of consecutive periods of time. Stages are usually identified through positive integers $1, 2, \dots$, and are characterized as follows:

Stages

- during each stage one or more stochastic (i.e. uncertain) events take place, and
- at the end of a stage decisions are taken, taking into account the specific outcomes of the stochastic events of this and previous stages.

Stochastic events may be such quantities as the demand realized during a period. They are usually represented as input data used in the deterministic model. Any variables in the deterministic model that are modeled conceptually to take their value *at the beginning* of a period (for instance, the stock at the beginning of a period), should be considered as decisions taken at the end of the previous period/stage within the stage concept.

If the deterministic model already contains a HORIZON (see Section 31.3) or any other set of time periods, the stages of the stochastic model may naturally coincide with the time periods from the deterministic model, but this certainly needs not be the case. A single period model, possibly even without an explicit period set, but with variables representing decisions taken at the beginning *and* at the end of the period, may still constitute a two-stage stochastic model. For a multi-period model, a single stage in the stochastic model may consist of multiple time periods from the deterministic model, and hence one can always construct a mapping from the deterministic period set to stages in the stochastic model.

Stages versus model periods

Every individual stochastic variable in a stochastic model should be uniquely associated with a single stage. This stage represents the period at the end of which the variable conceptually takes its value

Variables and stages

- taking into consideration the *specific* outcomes of stochastic events taking place during the stage at hand and during prior stages,
- but only taking into account the *distribution* of possible outcomes of the stochastic events taking place in any further stage.

Even if model periods of the deterministic model and the stages of the stochastic model coincide, variables with an index into the period set do not have to be associated with the stage corresponding to the value of that index. As discussed above, variables that conceptually take their value at the beginning of a

period, provide a first example of this behavior as they must be associated with the stage corresponding to the previous period within the stochastic model.

Other examples may arise, for instance, when the monthly productions of January, February and March should be decided upon prior to the beginning of January, regardless of the specific outcomes for the demands during these months. Conversely, if market research has delivered good estimates for the demand in January, February and March, the decisions for the production in these months should take into consideration the demand estimates of all three months. Hence the production variables for January, February and March should be part of the stage associated with March.

Examples

A *scenario* for a stochastic model is a collection of outcomes for all the stochastic events taking place in the model, along with the associated probability of the scenario to occur. For the event values associated with each scenario, one could solve a deterministic model, which would yield the optimal decisions for that particular scenario. For different scenarios, however, the decisions resulting from solving such deterministic models individually are, in general, completely unrelated, even if the event outcomes of the scenarios are exactly the same up to a certain stage. To address this problem, the scenarios of a stochastic model must be organized into a scenario tree.

Scenarios

A single scenario can be graphically represented as a simple tree illustrated in Figure 19.1.

Scenario trees

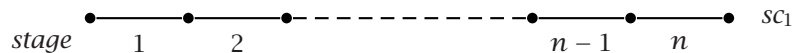


Figure 19.1: A tree representing a single scenario sc_1

For multiple scenarios, the specific outcomes of the stochastic events up to a certain stage usually coincide for a subset of the scenarios. This gives rise to a scenario tree as illustrated in Figure 19.2. In such a scenario tree, the path from the root node of the tree to each of its leaf nodes corresponds to a single scenario, and the event outcomes for scenarios that pass through the same intermediate node are identical for all stages up to that node. If a stage consists of multiple time periods in the deterministic model, this means that the stochastic events taking place during *all* periods associated with the stage should coincide. The solution process of a stochastic model will make sure that the decisions that are to be taken at the end of these stages are identical for all the scenarios passing through the node, as one would intuitively expect.

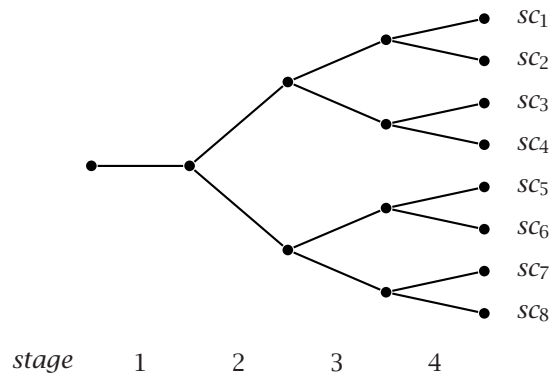


Figure 19.2: A scenario tree with 8 scenarios

The scenarios and the scenario tree used in a stochastic model are usually generated by using one of the following two techniques

Scenario generation

- generate scenarios by incrementally creating a scenario tree according to a given distribution for each stochastic event, or
- given an externally created set of scenarios, create a scenario tree by grouping identical or similar scenarios at every level of the tree.

Given a leaf node in an intermediate scenario tree, for every stochastic event that occurs during the stage directly following that node, a fixed number of values is computed according to a given distribution (each with its own relative probability of taking place). For each of these values a new branch is added to the node. The process starts by adding branches to the root node of the tree and ends when a tree is generated for all stages. The total number of scenarios generated by the process is the final number of leaf nodes generated. The probability of a scenario is the multiplication of the relative probabilities associated with each branch along the path from root to leaf node. The scenario tree in Figure 19.2 could be generated in this way, for example, by choosing, at every intermediate node, a *high* or a *low* level for the demand during the stage following that particular node.

Distribution-based scenario generation

Another approach is to start from a given collection of scenarios with probabilities adding up to 1. Such a collection of scenarios can either be randomly generated or be the result of some external process. As a tree, they can be represented as a trivial scenario tree, as illustrated in Figure 19.3. This tree can be transformed into a scenario tree by bundling together identical or similar scenarios into a fixed or dynamic number of branches. The group of scenarios passing through a particular intermediate node in the scenario tree is analyzed and grouped into subgroups of scenarios with similar or identical outcomes of the stochastic events during the stage following that node. For every subgroup, the existing branches are bundled into a single branch, and

Scenario-based tree generation

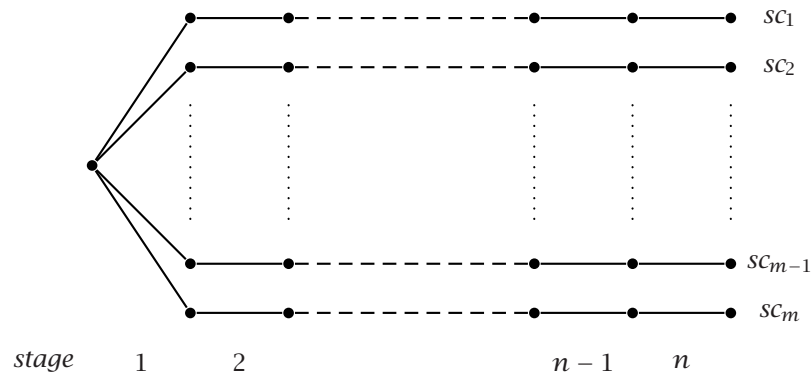


Figure 19.3: An initial disconnected scenario tree

the stochastic event outcomes are made identical for all scenarios in the subgroup. The process starts by analyzing all scenarios at the root node of the tree, and ends when every scenario is associated with a single leaf node.

The implementation of stochastic programming in AIMMS closely follows the concepts described in this section. The basic procedure to create and solve a stochastic model in AIMMS is as follows:

*Basic procedure
for solving
stochastic
models*

- indicate in your model which parameters and variables are to become stochastic,
- for every stochastic variable in your model specify during which stage of the stochastic model the decision is to be taken,
- generate scenarios, their stochastic data, and a scenario tree, using one of the techniques described above, and
- generate and solve the stochastic model using the special methods available for this purpose in AIMMS.

Each of these steps is explained in more detail in the sections to follow. Note that changing parameters and variables in your model into stochastic parameters and variables, does in no way influence the possibility to solve the underlying deterministic model in its original form. Thus, the stochastic programming facilities in AIMMS always form a true extension of the functionality of the existing AIMMS application.

19.2 Stochastic parameters and variables

To allow the storage of scenario-dependent parameter and variable data for multiple scenarios in a stochastic model, all such scenarios should be added to the predefined set `AllStochasticScenarios`. If your application contains multiple stochastic models—each with different scenario sets—the set `AllStochasticScenarios` should be the union of all these scenario sets. For each stochastic model you can then define an associated subset of `AllStochasticScenarios` to use with that particular stochastic model.

The set All-Stochastic-Scenarios

Stochastic events are modeled in AIMMS as numeric `PARAMETERS` for which the `Stochastic` property has been set (see also Section 4.1). For stochastic parameters AIMMS provides an additional `.Stochastic` suffix, which you can use to store scenario-dependent stochastic event outcomes. The data stored in the suffix is used by AIMMS when generating the stochastic model. The index domain of the `.Stochastic` suffix is, therefore, the set `AllStochasticScenarios` plus the original domain of the parameter.

Stochastic parameters

Consider the following declarations

Example

```
SET:
  identifier   : MyScenarios
  subset of   : AllStochasticScenarios
  index       : sc;

PARAMETER:
  identifier   : Demand
  index domain : (c,t)
  property    : Stochastic ;
```

These declarations will cause AIMMS to create a `.Stochastic` suffix for the parameter `Demand(c,t)`. To use, or assign values to, `Demand.Stochastic`, you must use an additional index into (a subset of) `AllStochasticScenarios`. The following statement provides an example of such a statement.

```
Demand.Stochastic(sc,c,t) := Uniform(10,20);
```

If a constraint contains a reference to the parameter `Demand`, AIMMS will use the data in `Demand.Stochastic` to generate the appropriate demand constraint for every scenario.

By setting the `Stochastic` property for a `VARIABLE` in your model, you indicate to AIMMS that this variable may have multiple, scenario-dependent, solutions when used in a stochastic model. Consequently, when generating a matrix for the stochastic model, a column will be generated conceptually for every single scenario.

Stochastic variables

For stochastic variables you must also specify the mandatory STAGE attribute. Through the STAGE attribute you specify the stage at the end of which the decision corresponding to the stochastic variable is to be taken. The value of the STAGE attribute must be an explicit positive integer value, or a parameter reference involving some or all of the indices on the index list of the declared variable.

The STAGE attribute

As discussed in the previous section, for every scenario s_0 , a stochastic variable x gets its value x_{s_0} at the end of stage n as specified in the STAGE attribute of the variable. In addition, its value is based on the specific outcomes of the stochastic events for that scenario taking place during stages $1, \dots, n$, but only on the distribution of the stochastic event outcomes for any further stages. Therefore, the value x_s must be equal to x_{s_0} for every other scenario s that passes through the same node in the scenario tree at the end of stage n as s_0 . The constraints enforcing this equality are called *non-anticipativity constraints*—they do not allow the solution to anticipate on stochastic outcomes that lie beyond the stage as specified by the STAGE suffix.

Non-anticipativity constraints...

When generating a stochastic model, AIMMS will automatically enforce the non-anticipativity constraints, either by explicitly adding them to the generated matrix, or implicitly by substituting a single representative x_{s_0} for every other variable x_s . While enforcing non-anticipativity in an implicit manner will drastically reduce the matrix size, an explicit representation may be helpful for solvers able to decompose the generated matrix.

... enforced explicitly or implicitly

If a variable in a stochastic model has not been declared stochastic, it is deterministic in the sense that it assumes the same value for every scenario, as is the case with first stage variables.

Non-stochastic variables

Variables can also have a .Stochastic suffix in AIMMS. It follows the same rules for its index domain as the .Stochastic suffix of parameters. AIMMS uses the .Stochastic suffix of variables to store the solution data of a stochastic model after solving it. However, contrary to stochastic parameters, AIMMS will not only create the .Stochastic suffix for stochastic variables, but for *all* variables that are involved in a stochastic model.

The .Stochastic suffix for variables

The values stored in the .Stochastic suffix after solving a stochastic model for each type of variable are as follows:

Contents of .Stochastic suffix

- for stochastic variables, the .Stochastic suffix will contain the solution of the variable for each scenario,
- for the objective variable, the .Stochastic suffix will contain the contribution to the objective of each scenario, as well as the weighted objective value of the stochastic model itself,

- for any other non-stochastic variable, the `.Stochastic` suffix will contain the deterministic solution of that variable for the stochastic model.

As the solution of a stochastic model is entirely stored in the `.Stochastic` suffix, the solution of the underlying deterministic model remains completely intact after solving the stochastic model. This makes it easy to visually, and/or programmatically, compare the solutions of the deterministic and stochastic model.

As the objective value and solution of the non-stochastic variables of the stochastic model cannot be coupled directly with one specific scenario in the scenario set, AIMMS creates an extra element in the set `AllStochasticScenarios` for this purpose. You must specify the name of this element when solving the stochastic model (see also Section 19.4).

*Non-stochastic
solution data*

19.3 Scenario generation

To support you in creating scenarios and a scenario tree, AIMMS provides a system module which provides a customizable scenario generation framework. For each of the two basic methods for scenario generation discussed in Section 19.1, the module contains a generic procedure to implement that method. To use these scenario generation procedures to generate scenarios and/or a scenario tree, you only have to implement some callback procedures to supply the necessary data for your specific stochastic model.

*Scenario
generation*

To import the generation module into your model, select **Install System Module...** from the **Settings** menu, and select the **Scenario Generation Module** from the dialog box that appears. The module will be added at the end of the model tree of your model. By default, the module prefix of the **Scenario Generation Module** is `ScenGen`.

*Importing the
system module*

19.3.1 Distribution-based scenario generation

The basic procedure in the scenario generation module for distribution-based scenario generation is

*Distribution-
based scenario
generation*

- `CreateScenarioTree(Stages, Scenarios, ScenarioProbability, ScenarioTreeMapping)`.

The procedure has a single input argument:

*Input
arguments*

- the set of *Stages* in your stochastic model. This set must be a subset of the predefined set `Integers`.

The outputs of this procedure are:

- the set of *Scenarios* (which must be a subset of `AllStochasticScenarios`) generated by the procedure,
- the *ScenarioProbability*, a one-dimensional parameter indexed over the set *Scenarios*, and
- the *ScenarioTreeMapping*, a two-dimensional element parameter defined over *Scenarios* × *Stages* to *Scenarios*, providing a mapping from every scenario during every stage to a single representative scenario for the scenario bundle in which the given scenario is contained during this stage.

Output arguments

The contents of the outputs of the procedure `CreateScenarioTree` is completely based on the results of the problem-specific callbacks that you have to supply. The following callbacks are expected by `CreateScenarioTree`:

Distribution-based callback functions

- `InitializeNewScenarioCallback(CurrentStage, Scenario, RepresentativeScenario)`,
- `InitializeStochasticDataCallback(CurrentStage, Scenario, ChildBranch, ChildBranchName)`, and
- `InitializeChildBranchesCallback(CurrentStage, Scenario, ChildBranches, ChildBranchNames)`.

When building up the scenario tree, AIMMS creates new scenarios on the fly. In order for you to refer to data from previous stages for this scenario, AIMMS will call the callback `InitializeNewScenarioCallback` for every *Stage* prior to the current stage, and supply the *RepresentativeScenario* from the scenario bundle for *CurrentStage* which also contains the newly created *Scenario*. By copying the stochastic data for this stage from this representative scenario, you make it available both to you and AIMMS. To properly generate the stochastic model, AIMMS needs the stochastic parameter values for every stage and every scenario.

Initializing a new scenario

In the procedure `InitializeStochasticDataCallback` you can provide values to all stochastic parameter values for the *ChildBranch* during *CurrentStage* for the *Scenario*. Because AIMMS has called the `InitializeNewScenarioCallback` prior to calling `InitializeStochasticDataCallback` you also have access to the stochastic parameter values of this scenario prior to the current stage. Based on the value of *ChildBranch* and the prior stochastic parameter values, you should have sufficient information to generate new stochastic parameter values for the current stage. You should pass the relative weight of this branch compared to the other child branches through the return value of the callback. Note that the relative weights you return may, but need not, add up to one. After creating scenarios for all branches, AIMMS will scale the sum of the returned relative weights of all branches to one.

Supplying stochastic event data

Finally, to extend the scenario tree to a next stage, AIMMS calls the callback `InitializeChildBranchesCallback`. In this callback, you should fill the Integer subset `ChildBranches` with integers 1,2,... for every child branch that you want to add to `Scenario` at `CurrentStage`. Through the element parameter `ChildBranchNames` you should provide a short representative name for every child branch (for instance, "H" and "L" when child branches represent high and low demand). From the branch names you supply, AIMMS will generate the full names of the final element names of the scenarios generated by the scenario generation procedure (for instance ' [H,L,H,H,L]' for a scenario with high, low, high, high, and low demand values during the successive stages of the scenario).

Generating new child branches

The scenario generation module contains templates for each of the callbacks described above. Rather than changing these template callbacks in the module, you are advised to copy the template callbacks to your core model, and change the bodies of the copied callbacks. Finally, you should notify AIMMS of the names of your callback functions by, prior to calling the procedure `CreateScenarioTree`, assigning the names of your callback procedures to the element parameters

Setting the callbacks

- `ScenGen::InitializeNewScenarioCallbackFunction`,
- `ScenGen::InitializeStochasticDataCallbackFunction`, and
- `ScenGen::InitializeChildBranchesCallbackFunction`.

The following callbacks will cause the procedure `CreateScenarioTree` to generate a tree with 2 branches "H" and "L" (for high and low demand) at every intermediate node, and initialize `Demand.Stochastic` for every period. The example assumes the existence of a mapping `PeriodToStage(st,t)`.

Example

To initialize a new scenario, we have to copy the stochastic demand data for the newly created `Scenario` during `Stage` from the scenario `RepresentativeScenario`. Thus, the body of the `InitializeNewScenarioCallback` would read

Initializing a new scenario

```
for ( t | PeriodToStage(CurrentStage,t) ) do
  Demand.Stochastic(Scenario,t) := Demand.Stochastic(RepresentativeScenario,t);
endfor;
```

To generate two child branches to any intermediate node in the scenario tree representing high ("H") and low ("L") demand, the implementation of the `InitializeChildBackBranchesCallback` should be

Generating new child branches

```
ChildBranches := { 1, 2 };
ChildBranchNames('1') := "H";
ChildBranchNames('2') := "L";
```

For each newly added child branches, the following implementation of `InitializeStochasticDataCallback` assigns a high (20) or low (10) stochastic demand value to the *Scenario* during the *CurrentStage*

Initializing stochastic demand

```
for ( t | PeriodToStage(CurrentStage,t) ) do
  Demand.Stochastic(Scenario,t) := if ( ChildBranch = 1 ) then 20 else 10 endif;
endfor;

return 1;
```

By returning 1 for all branches, we just indicate that every branch has equal relative weight. For two branches, this will result in a relative probability for each branch of 0.5.

19.3.2 Scenario-based tree generation

The basic procedure in the scenario generation module for scenario-based tree generation is

Scenario-based tree generation

- `CreateScenarioData(Stages, Scenarios, ScenarioProbability, ScenarioTreeMapping)`.

The procedure has a single input argument:

Input arguments

- the set of *Stages* in your stochastic model. This set must be a subset of the predefined set `Integers`.

The outputs of the procedure are:

Output arguments

- the set of *Scenarios* for which you have provided stochastic parameter values,
- the *ScenarioProbability*, a one-dimensional parameter indexed over the set *Scenarios*, and
- the *ScenarioTreeMapping*, a two-dimensional element parameter defined over $Scenarios \times Stages$ to *Scenarios*, providing a mapping from every scenario during every stage to a single representative scenario for the scenario bundle in which the given scenario is contained during this stage.

The procedure `CreateScenarioData` will help you construct a scenario tree as follows:

Algorithm outline

- initially, AIMMS will request you to generate a set of scenarios with their relative weights,
- next, AIMMS will ask you, to divide a given group of scenarios at the current stage into a number of subgroups of equal or similar scenarios at the next stage,

- AIMMS will request you to reassign a single unique value to each stochastic event parameter for all scenarios in a scenario group (e.g. the mean over all scenarios in the group), and
- finally, AIMMS will remove scenarios which you identify as identical.

For each of the steps outlined in the previous paragraph, you must supply a callback procedure:

Scenario-based callbacks

- `InitializeStochasticScenarioDataCallback(Scenario, Scenarios)`,
- `DetermineScenarioGroupsCallback(CurrentStage, ScenarioGroup, ScenarioGroupOrder)`,
- `AssignStochasticDataForScenarioGroupCallback(CurrentStage, ScenarioGroup)`, and
- `CompareScenariosCallback(Scenario1, Scenario2, Stages, FirstDifferentStage)`

Through the `InitializeStochasticScenarioDataCallback` you must supply the stochastic event data during all stages for a *Scenario* generated by AIMMS. The function should return the relative weight of the scenario compared to all other scenarios you supply. If you are done adding scenarios, the callback should return the value 0.

Initializing scenarios

If you have already read scenario data from a database, for instance, you can overwrite the generated value of *Scenario* argument with an existing scenario name read from the database. In that case, if you have read the stochastic data directly into the `.Stochastic` suffix of the stochastic parameters in your model, you only have to return the relative weight.

Dealing with existing scenario data

If you do not have existing scenario data, you should generate stochastic data for the *Scenario* element generated by AIMMS for all stochastic parameters in your model. If you want to change the name of the generated *Scenario*, you can do so using the function `SetElementRename`.

Supplying new scenario data

In the `DetermineScenarioGroupsCallback`, you must divide the scenarios in *ScenarioGroup* created during a previous stage (or the group of all scenarios to start with during the first stage) into subgroups, based on the equality or similarity of the stochastic event values associated with the scenarios during *CurrentStage*. You must specify the subgroups by assigning a *ScenarioGroupOrder* to every scenario in the *ScenarioGroup*, where scenarios with the same assigned order form a subgroup during the current stage.

Creating scenario subgroups

If the stochastic event parameters in *ScenarioGroup* during *CurrentStage* are similar, but not equal, you must make sure to assign identical event parameter values to every scenario when AIMMS calls the *AssignStochasticDataForScenarioGroupCallback*. Failure to do so may result in infeasible stochastic models generated by AIMMS.

Assigning stochastic event values

Finally, AIMMS will probe for identical scenarios through the *CompareScenarioCallback*, remove duplicate scenarios when encountered, and adjust the scenario probabilities accordingly. When the stochastic event values of *Scenario1* and *Scenario2* are identical during *Stages*, the callback should return 0. If the scenarios are not identical the callback should have a nonzero return value, and set the output argument *FirstDifferentStage* equal to the first stage during which the event parameters differ for both scenarios.

Removing identical scenarios

The scenario generation module contains templates for each of the callbacks described above. Rather than changing these template callbacks in the module, you are advised to copy the template callbacks to your core model, and change the bodies of the copied callbacks. Finally, you should notify AIMMS of the names of your callback functions by, prior to calling the procedure *CreateScenarioData*, assigning the names of your callback procedures to the element parameters

Setting the callbacks

- *ScenGen::InitializeStochasticScenarioDataCallbackFunction*,
- *ScenGen::DetermineScenarioGroupsCallbackFunction*,
- *ScenGen::AssignStochasticDataForScenarioGroupCallbackFunction*, and
- *ScenGen::CompareScenariosCallbackFunction*.

The callbacks for scenario-based tree generation, are usually more problem-specific, and hence less instructive, than the callbacks for the tree-based scenario generation scheme. Therefore, rather than including a lengthy example here, we refer to the example models for stochastic programming that come with your AIMMS system.

Example

The scenario generation module is completely implemented in the AIMMS language itself, and contains basic implementations of both scenario generation methods, which will provide a good starting point for most stochastic models. If neither of these implementations fits your needs, you can copy the module to your project directory, replace the system module with the copy, and make the algorithms in the copied module more advanced to better fit the needs of your stochastic model.

Scenario generation can be modified

19.4 Solving stochastic models

After generating stochastic event data and a scenario tree, you can generate and solve the stochastic model by using methods from the GMP library discussed in Chapter 21. AIMMS supports two methods for solving a stochastic model:

Solving stochastic models

- by solving its *deterministic equivalent*, or
- for purely linear mathematical programs only, through the *stochastic Benders algorithm*.

The latter algorithm will decompose the stochastic model into multiple smaller models, and thus is better suited to solve stochastic models where the deterministic equivalent, either by the size of the deterministic model or because of a huge number of scenarios, becomes too big or time-consuming to solve at once.

19.4.1 Generating and solving the deterministic equivalent

The method for generating a stochastic model for a MATHEMATICAL PROGRAM *MP* is

Generating a stochastic model

- `GMP::Instance::GenerateStochasticProgram(
MP, StochasticParameters, StochasticVariables,
Scenarios, ScenarioProbability, ScenarioTreeMap,
DeterministicScenarioName[, GenerationMode][, Name)`

The function returns an element into the set `AllGeneratedMathematicalPrograms`. This generated math program instance contains a memory-efficient representation of the technology matrix of the stochastic model and the stochastic event data, and can be used to create a deterministic equivalent of the stochastic model, as well as the submodels necessary for a stochastic Benders approach.

Through the arguments *StochasticParameters* and *StochasticVariables* you indicate to AIMMS which stochastic parameters and variables you want to take into consideration when generating this stochastic model. These arguments must be subsets of the predefined sets `AllStochasticParameters` and `AllStochasticVariables`, respectively. You may want to use real subsets, for instance, when your AIMMS project contains multiple stochastic models, each referring only to a subset of the stochastic parameters and variables.

Specifying stochastic identifiers

Through the *Scenarios*, *ScenarioProbability* and *ScenarioTreeMap* arguments you specify the set of scenarios, their probabilities and the mapping defining the scenario tree for which you want to generate the stochastic model to AIMMS. Through the string argument *DeterministicScenarioName*, you supply the name of the artificial element that AIMMS will add to the predefined set *AllStochasticScenarios* (if not created already), and use to store the solution of non-stochastic variables in their respective *.Stochastic* suffices as explained in Section 19.2.

Specifying scenarios

Using the *GenerationMode* argument you can specify whether you want AIMMS to explicitly add the non-anticipativity constraints to your stochastic model, or whether you want non-anticipativity to be enforced implicitly by substituting the representative scenario for every non-representative scenario at every stage. *GenerationMode* is an element parameter into the predefined set *AllStochasticGenerationModes*, with possible values

Enforcing non-anticipativity constraints

- 'CreateNonAnticipativityConstraints', and
- 'SubstituteStochasticVariables' (the default value).

With the optional *Name* argument you can explicitly specify a name for the generated mathematical program. If you do not choose a name, AIMMS will use the name of the underlying MATHEMATICAL PROGRAM as the name of the generated mathematical program as well. Please note, that AIMMS will also use this name as the default name for solving the deterministic model. Therefore, if you do not want the generated mathematical program of the deterministic model to be deleted, then you have to choose a non-default name.

Name argument

You can solve a stochastic model by using the regular GMP procedure

- `GMP::Instance::Solve(gmp)`

Solving the deterministic equivalent of a stochastic model

By applying this function to a generated mathematical program associated with a stochastic model, AIMMS will create the deterministic equivalent and pass it to the appropriate LP/MIP solver. The `GMP::Instance::Solve` method is discussed in full detail in Section 21.2.

Note that, when you adjust the scenario tree map, the stochastic data, the scenario probabilities, or the value of the *STAGE* attribute of some variables after you generated the stochastic model, you should regenerate the stochastic model again to reflect these changes.

Changing the model input

Consider the following call to `GMP::Instance::GenerateStochasticProgram`

Example

```
GMP::Instance::GenerateStochasticProgram(
  TransportModel, AllStochasticParameters, AllStochasticVariables,
  MyScenarios, MyScenarioProbability, MyScenarioTreeMap,
  "TransportModel", 'SubstituteStochasticVariables', "StochasticTransportModel");
```

After solving the generated stochastic model, its solution will be stored as follows, where `sc` is an index into `MyScenarios`

- the per-scenario solution of a stochastic variable `Transport(i,j)` will be stored in `Transport.Stochastic(sc,i,j)`,
- the deterministic solution of a non-stochastic variable `InitialStock(i)` will be stored in `InitialStock.Stochastic('TransportModel',i)`,
- the weighted objective value for the objective variable `TotalCost` will be stored in `TotalObjective.Stochastic('TransportModel')`, while the contribution by every scenario is available through `TotalCost.Stochastic(sc)`.

19.4.2 Using the stochastic Benders algorithm

Instead of solving the deterministic equivalent of a stochastic model, AIMMS also allows you to solve *linear* stochastic models using a stochastic Benders algorithm. The stochastic Benders algorithm is based on a reformulation of the original model as a sequence of models outlined below. The solution of the original model can be achieved by solving the sequence of models iteratively until a terminating condition is reached. A more detailed discussion of the stochastic Benders algorithm can be found in [De98] or [Al03].

Using the stochastic Benders algorithm

All nodes in the scenario tree are numbered starting at 1 (the root node).

Definitions

Indices:

i *index for the set of nodes N*
 t *index for the set of stages T*

Parameters:

q_i *probability belonging to node i*
 p_i *parent of node i*

Sets:

I_i *set with children of node i*
 N_t *set of nodes belonging to stage t*

In the algorithmic outline below we identify the problem names with their associated solutions. That is, if a problem is, for instance, identified as $F_i(x_{p_i})$, we will also use this name to denote its solution in other sub-problems.

Convention

The nested Benders algorithm can be used for problems of the form

The original model

Minimize:

$$\sum_{t \in T} \sum_{i \in N_t} q_i c_i^T x_i$$

Subject to:

$$\begin{aligned} W_1 x_1 &= h_1 \\ A_i x_{p_i} + W_i x_i &= h_i \quad \forall i \in N_t, t \in T \setminus \{1\} \\ x_i &\geq 0 \quad \forall i \in N_t, t \in T \end{aligned}$$

This problem corresponds to the following sequence of problems. For node $i = 1$, the problem F_1 is defined as

*A reformulation
as a sequence of
models*

Minimize:

$$c_1^T x_1 + \sum_{j \in I_1} q_j F_j(x_1)$$

Subject to:

$$\begin{aligned} W_1 x_1 &= h_1 \\ x_1 &\geq 0 \end{aligned}$$

For all other nodes $i \in N_t$ in stage $t \in T \setminus \{1\}$, the problem $F_i(x_{p_i})$ is defined as follows (note that $\sum_{j \in I_i} q_j = q_i$)

Minimize:

$$c_i^T x_i + \sum_{j \in I_i} \frac{q_j}{q_i} F_j(x_i)$$

Subject to:

$$\begin{aligned} W_i x_i &= h_i - A_i x_{p_i} \\ x_i &\geq 0 \end{aligned}$$

For the leaf nodes in the scenario tree, the term $\sum_{j \in I_i} \frac{q_j}{q_i} F_j(x_i)$ is omitted.

If we now introduce an upper bound θ_i to replace the term $\sum_{j \in I_i} \frac{q_j}{q_i} F_j(x_i)$, we can rewrite the subproblem $F_i(x_{p_i})$ as

*Formulated
differently*

Minimize:

$$c_i^T x_i + \theta_i$$

Subject to:

$$\begin{aligned} W_i x_i &= h_i - A_i x_{p_i} \\ \theta_i &\geq \sum_{j \in I_i} \frac{q_j}{q_i} F_j(x_i) \\ x_i &\geq 0 \end{aligned}$$

Because of the linear nature of the original problem, the terms $\sum_{j \in I_i} \frac{q_j}{q_i} F_j(x_i)$ are piecewise linear and convex. Therefore there exists an (a priori unknown) set of equations

$$D_i^l x_i = d_i^l$$

that describes such a term and for which

$$D_i^l x_i + \theta_i \geq d_i^l.$$

Moreover, we are only interested in those x_i such that $F_j(x_i)$ are feasible for all $j \in I_i$. This requirement can be enforced by an (a priori unknown) set of constraints

$$E_i^l x_i \geq e_i^l.$$

By substituting these constraints we obtain the following reformulation of problem $F_i(x_{p_i})$

Minimize:

$$c_i^T x_i + \theta_i$$

Subject to:

$$\begin{aligned} W_i x_i &= h_i - A_i x_{p_i} \\ D_i^l x_i + \theta_i &\geq d_i^l && \forall l \in 1, \dots, R_i \\ E_i^l x_i &\geq e_i^l && \forall l \in 1, \dots, S_i \\ x_i &\geq 0 \end{aligned}$$

The actual problem that is solved at node i is the following relaxed master problem $\tilde{N}_i(x_{p_i})$ defined as follows:

The relaxed master problem

Minimize:

$$c_i^T x_i + \theta_i$$

Subject to:

$$\begin{aligned} W_i x_i &= h_i - A_i x_{p_i} \\ D_i^l x_i + \theta_i &\geq d_i^l && \forall l \in 1, \dots, r_i \\ E_i^l x_i &\geq e_i^l && \forall l \in 1, \dots, s_i \\ x_i &\geq 0 \end{aligned}$$

At the start of the Benders algorithm r_i and s_i will be 0 for all $i \in N$. The constraints $D_i^l x_i + \theta_i \geq d_i^l$ are optimality cuts obtained from the children. That is, if $\tilde{N}_j(x_i)$ is feasible for all $j \in I_i$ (but not optimal) then an optimality cut is added to $\tilde{N}_i(x_{p_i})$. The optimality cut is constructed by using a combination of the dual solutions of $\tilde{N}_j(x_i)$ for all $j \in I_i$. Adding an optimality cut does not make a feasible relaxed master problem infeasible. The Benders algorithm fails if one of the subproblems is unbounded. This can be avoided by giving all variables, except the objective variable, finite bounds.

The constraints $E_i^l x_i \geq e_i^l$ are feasibility cuts obtained from a child. If some child problem $\tilde{N}_j(x_i)$ is not feasible then the following problem $\tilde{E}_j(x_i)$ is solved

*Adding
feasibility cuts*

Minimize:

$$w_j = e^T u_j^+ + e^T u_j^-$$

Subject to:

$$\begin{aligned} W_j x_j + I u_j^+ - I u_j^- &= h_j - A_j x_i \\ E_j^l x_j &\geq e_j^l && \forall l \in 1, \dots, s_j \\ x_j &\geq 0 \\ u_j^+ &\geq 0 \\ u_j^- &\geq 0 \end{aligned}$$

This feasibility problem can only be formulated for linear problems, is always feasible, and bounded from below by 0. Its dual solution is used to construct a new feasibility constraint for $\tilde{N}_i(x_{p_i})$. Note that node j in its turn obtains optimality and/or feasibility cuts from its children for $\tilde{N}_j(x_i)$ and $\tilde{E}_j(x_i)$, unless j refers to a leaf node.

If (x_i, θ_i) is an optimal solution of $\tilde{N}_i(x_{p_i})$ and

$$\theta_i \geq \tilde{N}_i(x_{p_i})$$

*Terminating
condition*

then (x_i, θ_i) is an optimal solution of $F_i(x_{p_i})$. If this holds for all non-leaf nodes then we have found an optimal solution of our original problem. For the leaf nodes, x_i only needs to be an optimal solution of $\tilde{N}_i(x_{p_i})$.

The stochastic Benders algorithm outlined above is implemented in AIMMS as a system module that you can include into your model, together with a number of supporting functions in the GMP library to perform a number of algorithmic steps that cannot be performed in the AIMMS language itself, for instance, to actually generate the stochastic sub-problems, and to generate feasibility and optimality cuts.

*Implementation
in AIMMS*

You can add the system module implementing the stochastic Benders algorithm to your model through the **Settings-Install System Module...** menu. By selecting the **Stochastic Decomposition Module** in the **Install System Module** dialog box, AIMMS will add this system module to your model.

*Adding the
module*

The main procedure for using the stochastic Benders algorithm is `DoStochasticDecomposition`. Its inputs are:

- a stochastic GMP,
- the set of stages to consider, and
- the set of scenarios to consider.

The procedure implements the algorithm outlined above. The supporting GMP functions for the stochastic Benders algorithm are described in Section 21.7.

Because the stochastic Benders algorithm is written in the AIMMS language, you have complete freedom to modify the algorithm in order to tune it for your stochastic programs. Also, the basic algorithm solves all sub-problems sequentially. If your AIMMS license permits parallel solver sessions, you can also speed up the algorithm by solving multiple sub-problems in parallel using the GMP function `GMP::SolverSession::AsynchronousExecute`.

*Using the
stochastic
Benders module*

*Modifying the
algorithm*

Bibliography

- [Al03] F. Altenstedt, *Memory consumption versus computational time in nested benders decomposition for stochastic linear programmings*, Tech. report, Chalmers University of Technology, Göteborg, Sweden, 2003.
- [Bi97] J.R. Birge and F. Louveaux, *Introduction to stochastic programming*, Springer, New York, 1997.
- [De98] M. Dempster and R. Thompson, *Parallelization and aggregation of nested benders decomposition*, *Annals of Operations Research* **81** (1998), 163–187.
- [Ka05] P. Kall and J. Mayer, *Stochastic linear programming: Models, theory, and computation*, Springer, New York, 2005.