# Chapter 1
# Stochastic Programming from Modeling Languages

Emmanuel Fragnière[1] and Jacek Gondzio[2]

## 1.1 Introduction

The majority of deterministic mathematical programming problems have a compact formulation in terms of algebraic equations. Therefore they can easily take advantage of the facilities offered by algebraic modeling languages. These tools allow expressing models by using convenient mathematical notation (algebraic equations) and translate the models into a form understandable by the solvers for mathematical programs.

Algebraic modeling languages provide facility for the management of a mathematical model and its data, and access different general-purpose solvers. The use of algebraic modeling languages (AMLs) simplifies the process of building the prototype model and in some cases makes it possible to create and maintain even the production version of the model.

As presented in other chapters of this book, stochastic programming (SP) is needed when exogenous parameters of the mathematical programming problem are random. Dealing with stochasticities in planning is not an easy task. In a standard scenario-by-scenario analysis, the system is optimized for each scenario separately. Varying the scenario hypotheses we can observe the different optimal responses of the system and delineate the "strong trends" of the future. Indeed, this scenario-by-scenario approach implicitly assumes perfect foresight. The method provides a first-stage decision, which is valid only for the scenario under consideration. Having as many decisions as there are scenarios leaves the decision-maker without a clear recommendation. In stochastic programming the whole set of scenarios is combined into an event tree, which describes the unfolding of uncertainties over the period of planning. The model takes into account the uncertainties characterizing the scenarios through stochastic programming techniques. This adaptive plan is much closer, in spirit, to the way that decision-makers have to deal with uncertain future

in real life.

Most of the difficulties to model uncertainty through stochastic programming originate from the lack of an agreed standard of its representation. Indeed, stochastic programming problems usually involve dynamic aspects of decision making which combined with uncertainty inevitably leads to a complicated model. To make the problem tractable, uncertainty is usually expressed in terms of an approximate discrete distribution. However, the need of accuracy in modeling inevitably leads to the explosion of dimension in the size of the corresponding mathematical program. This imposes additional limits on the way of modeling stochastic programming problems and further complicates the management of such models. In consequence there still does not exist a *standard* way of modeling stochastic programming problems in algebraic modeling languages. However, AML developers are working on them and have already come up with a number of possible extensions.

In this chapter we address the difficulties of modeling stochastic programs and discuss in detail different approaches developed so far to deal with this problem.

The chapter is organized as follows. In Section 1.2 we briefly explain the important role played by AMLs in the development of optimization based models. In Section 1.3 we present different formulations of stochastic programs. In Section 1.4 we discuss specific issues related to an automatic generation of stochastic programs that result in difficulties with standardization of their generation by AMLs. In Section 1.5 we discuss the techniques of stochastic programming available to AMLs and in Section 1.6 we comment on the crucial issues of communication between the solver and the algebraic modeling language. Finally, in Section 1.7 we give our conclusions.

## 1.2 Algebraic Modeling Languages

Algebraic Modeling Languages (AMLs for short) enable decision models to be formulated with an algebraic notation. They use a generic model description in form of a data file. The models developed with AMLs can be easily modified. The user builds the model and provides the AML with the appropriate data. The AML translates the model into a form that is understandable to a solver and invokes the appropriate solver. In this setting, the solver is seen as a black box. The optimization code may query the AML about any additional information on the problem. For example, nonlinear optimization code may ask for the function values as well as the first and the second derivatives at a given point. Once the solution of the mathematical program is found, it is returned to the AML and the results are reported to the user.

AML enables a modeler to express the problem in an *index-based* mathematical form with abstract entities: sets, indices, parameters, variables and constraints. The key notion in the AML is the ability to group conceptually similar entities into a set. Once the entities are grouped in a given set, they can be referenced by indices to the elements of this set. This leads to a *problem formulation* that is very close to the formulation using algebraic notations. For instance, the mathematical operation $\sum_{i \in I} X_i$ is represented by the expression `SUM(I, X(I))` in the GAMS modeling

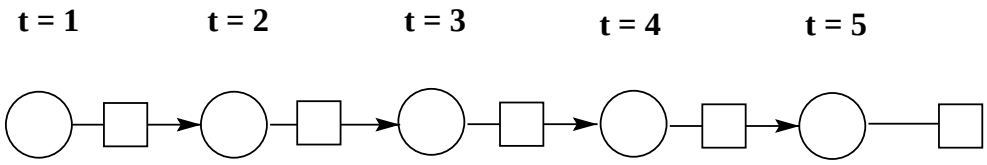



**Figure 1.1.** *Deterministic Invendeman model.*

language. The role of the AML is to expand the compact problem formulation (problem structure and data) into the *problem instance* which is ready to be solved by an appropriate optimization code. This operation is realized within the AML by replicating every entity over the different elements of the set. This is often referred to as a *set-indexing* ability of the AML. The user of an AML can define generic expressions that are indexed over several sets. Set-indexing in such cases involves *compound sets*.

There exist many algebraic modeling languages or more generally, optimization modeling languages (Fragnière and Gondzio 2002). Algebraic modeling languages such as GAMS (Brooke, Kendrick, and Meeraus 1992), AMPL (Fourer, Gay, and Kernighan 1993) or AIMMS (Bisschop and Entriken 1993) are routinely used by the mathematical programming community.

To illustrate the use of such modeling tools, we first present the algebraic formulation of a multiperiod inventory model with deterministic demands. A full description of this model, called *Invendeman*, can be found in Chapter 10 of the book by Thompson (1992). The model has the form of a simple optimization problem:

$$\begin{array}{ll} \max & \sum_{t=1}^{T}((p_t-2)x_t^- - (p_t+2)x_t^+ - hI_t) \\ \text{s.t.} & x_t^- - x_t^+ + I_t - I_{t-1} = -d_t \\ & I_t \le \bar{I} \\ & x_t^-, x_t^+, I_t \ge 0. \end{array} \tag{1.1}$$

The objective function corresponds to the net profit. There are three generic variables (inventory, quantity bought, quantity sold) and one generic constraint (inventory balance), all indexed over time. A representation of a five-period instance ($T = 5$) is shown in Figure 1.1. The variables used in the model have the following meaning:

- $t$ is the time period, $t = 1, 2, ..., T$,
- $T$ is the total number of time periods,
- $x_t^+$ is the quantity bought in period $t$,
- $x_t^-$ is the quantity sold in period $t$,
- 2 is the unit transactions cost which has to be paid each time a purchase or sale is made,
- $p_t$ is the market price at time $t$; a seller gets $p_t - 2$; a buyer pays $p_t + 2$,
- $d_t$ is the demand of the firm for the commodity at time $t$,
- $I_0$ is the initial stock of the commodity,

- $I_t$ is the stock of inventory held at time $t$,
- $I_T$ is the required final inventory of the commodity,
- $\bar{I}$ is the fixed warehouse capacity,
- $h$ is the unit holding cost for inventory.

We present below an extract of the corresponding model written using the GAMS (Brooke, Kendrick, and Meeraus 1992) modeling language (the full model along with the data can be found in Appendix .1).

```
OBJECTIVE..   PROFIT =E= SUM(INDEX,(P(INDEX)-2.0)*XMINUS(INDEX)
              -(P(INDEX)+2.0)*XPLUS(INDEX)-H*I(INDEX-1));
INVBAL(T-1).. XMINUS(T)-XPLUS(T)+I(T)-I(T-1) =E= -D(T);
```

We note that the formulation in GAMS is very close to the original algebraic formulation. In general terms, algebraic modeling languages provide declarative statements (as opposed to programming languages which contain procedural statements such as `loops` or `if-then-else` commands). This means that the code in the case of an AML can be seen as a declaration of the properties of the optimization problem.

The modeling language takes as an input, the algebraic formulation of the model and a set of data. Next all operations are automated. The modeling language generates a mathematical program, also called an instance of the problem. In a particular case of the deterministic multiperiod inventory model, the generated instance can be seen as a unique scenario. Later in this chapter we shall extend this model to take uncertainty into account. This will necessitate considering several scenarios.

## 1.3 Different Formulations of Stochastic Programming Problems

A multistage stochastic program with recourse is a multi-period mathematical program where parameters are assumed to be uncertain along the time path. The term *recourse* means that the decision variables adapt to the different outcomes of random parameters at each time period. A natural formulation of the stochastic programming problem relies on recursion (Birge et al. 1987) to describe dynamics of the modeled process. Several different formulations of SPs have been discussed in detail in Part 1 of the book. Therefore we omit recursive formulations and only briefly mention event trees and the deterministic equivalent formulation and an alternative formulation with nonanticipativity constraints, two forms which are most often used for modeling SPs using AMLs.

### 1.3.1 Event Tree and the Deterministic Equivalent Formulation

In a planning approach the evolution of uncertainties can be described as an alternation of decisions and random realizations. In its simplest form the discrete stochastic process can be represented as an event tree describing the unfolding of

the uncertainty over the period of planning (see Figure 1.2). A path, from the root to a leaf of the event tree, represents a scenario. Each scenario has a given probability. At each node of the event tree, a set of constraints and an objective function are defined. This involves variables specific to that node and its predecessor nodes. For instance, node 1 of the tree presented in Figure 1.2 corresponds to the first stage and associated decisions are identical for scenarios 1, 2, 3 and 4. At stage 2, decisions of scenarios 1 and 2 are identical. In the same way decisions of scenarios 3 and 4 are identical.

To formulate the deterministic equivalent of the multi-stage stochastic programming problem we first need to enumerate all nodes of the event tree (see Figure 1.2). We use a breadth-first search order, i.e., we start from a root node corresponding to the initial stage and end with leaf nodes corresponding to the last stage. Let $t = 1, 2, ..., T$ denote the stage and $l_t$ be the index of a node at stage $t$. Thus the root node has index $l_1 = 1$ and the stage 2 nodes start from index 2. Let $L_t$ denote the last node at stage $t$. Hence the nodes that belong to stage $t > 1$ have indices $l_t = L_{t-1}+1, L_{t-1}+2, \ldots, L_t$. To capture dynamics in the model we use $a(l_t)$ to denote the direct ancestor of node $l_t$. Clearly, the ancestor of $l_t$ is a node that belongs to stage $t-1$ (e.g., node 2 in Figure 1.2 is the ancestor of nodes 4 and 5). All decision variables $x$ are indexed only by the node number in the event tree: we use a superscript $l_t$. The stage the variable belongs to is therefore defined implicitly. The main constraint that describes system dynamics has the form

$$T^{l_t} x^{a(l_t)} + W^t x^{l_t} = h^{l_t}, \qquad l_t = L_{t-1}+1, L_{t-1}+2, \ldots, L_t, \tag{1.2}$$

where $T^{l_t}$ is the technology matrix that varies with the node in the event tree and $W^t$ is the recourse matrix that varies only with time but does not depend, in our example, on the realization within the same stage. One could obviously impose different conditions on matrices $T$ and $W$ and use indexing with time $t$ or both time and uncertainty $l_t$.

The deterministic equivalent formulation of the multi-stage problem has the following form

$$\begin{aligned}
\min\ & c'x^1 + \sum_{l_2=2}^{L_2} p^{l_2} (q^{l_2})' x^{l_2} + \sum_{l_3=L_2+1}^{L_3} p^{l_3} (q^{l_3})' x^{l_3} + \ldots + \sum_{l_T=L_{T-1}+1}^{L_T} p^{l_T} (q^{l_T})' x^{l_T} \\
\text{s.t. } & Ax^1 = b, \\
& T^{l_2} x^1 + W^2 x^{l_2} = h^{l_2},\ l_2 = 2, \ldots, L_2, \\
& T^{l_3} x^{a(l_3)} + W^3 x^{l_3} = h^{l_3},\ l_3 = L_2+1, \ldots, L_3, \\
& \qquad\qquad \ddots \qquad\quad \vdots \\
& T^{l_T} x^{a(l_T)} + W^T x^{l_T} = h^{l_T},\ l_T = L_{T-1}+1, \ldots, L_T, \\
& x^{l_t} \geq 0, \qquad\qquad l_t = 1, \ldots, L_T.
\end{aligned} \tag{1.3}$$

The numbers of children of each node in the event tree may differ as they depend on a probability distribution of the appropriate stochastic process. If the depth-first-search ordering of the nodes in the event tree is maintained during the generation of the mathematical program, the corresponding constraint matrix dis-

plays a nested dual block-angular structure. Links between the nested dual block-angular structure and the algebraic formulation of the original model can be easily established.

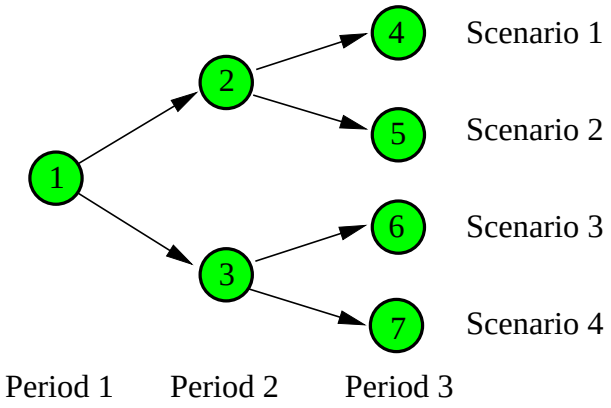



**Figure 1.2.** *A simple event tree.*

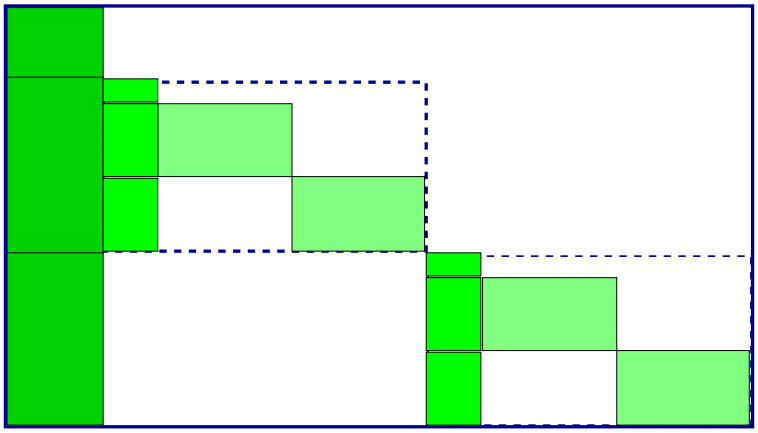


**Figure 1.3.** *The constraint matrix associated with the event tree.*

In our example (see Figure 1.2), node 1 is the root node, nodes 2 and 3 belong to stage 2 ($l_2 = 2, 3$), nodes 4 to 7 belong to stage 3 ($l_3 = 4, \ldots, 7$). The deterministic equivalent formulation of the problem is presented in (1.4). Let us observe that by shifting $x^3$ just after $x^5$ and shifting the third constraint after the fifth one, we immediately retrieve the structure presented in Figure 1.3. It is worth noting that this reordering operation means changing the breadth-first-search order of nodes in Figure 1.2 $(1, 2, 3, 4, 5, 6, 7)$ to the depth-first-search order $1, 2, 4, 5, 3, 6, 7$.

$$
\begin{array}{lll}
\min & c'x^1+p^2(q^2)'x^2+p^3(q^3)'x^3+p^4(q^4)'x^4+p^5(q^5)'x^5+p^6(q^6)'x^6+p^7(q^7)'x^7 & \\
\text{s.t.} & & \\
 & Ax^1 = b & \\
 & T^2x^1 + W^2x^2 = h^2 & \\
 & T^3x^1 + W^2x^3 = h^3 & \\
 & T^4x^2 + W^3x^4 = h^4 & \\
 & T^5x^2 + W^3x^5 = h^5 & (1.4)\\
 & T^6x^3 + W^3x^6 = h^6 & \\
 & T^7x^3 + W^3x^7 = h^7 & \\
 & x^1 \geq 0, \quad x^2 \geq 0, \quad x^3 \geq 0, \quad x^4 \geq 0, \quad x^5 \geq 0, \quad x^6 \geq 0, \quad x^7 \geq 0. &
\end{array}
$$

Note also that the probabilities in the objective function of problem (1.4) are not scenario probabilities but (partial) path probabilities: $p^n$ is the probability (at the start) that a path goes through node $n$. Clearly, (1.4) represents a structured linear program. Its structure should be exploited in the solution algorithm. Unfortunately, if the model is written with an algebraic modeling language, the structure, easily identifiable in the algebraic formulation, is usually lost when the corresponding mathematical program is sent to the solver. Each algebraic modeling language uses its own algorithm to generate an equivalent mathematical program, which scrambles the structure.

### 1.3.2 Formulation with Nonanticipativity Constraints

Another way to write the deterministic equivalent consists in creating independent copies of variables corresponding to every ancestor in the tree for every child of this node. In other words, we replicate the variable $x^{a(l_t)}$ in (1.2) and create copies $x^{l_t}_{t-1}$ for each $l_t$ corresponding to a child node of $a(l_t)$ in the event tree. We slightly change the notation at this point and add explicitly the stage subscript to each variable. Namely, with a given node $l_t$ at stage $t$ we associate two variables: an appropriate decision variable $x^{l_t}_t$ (at stage $t$) and a copy of the decision variable at the ancestor node corresponding to this particular child, $x^{l_t}_{t-1}$. For example, the variable $x^3$ representing the state corresponding to node 3 in stage 2 in Figure 1.2 would have two copies $x^6_2$ and $x^7_2$. Hence the last two constraints in (1.4) can be replaced with the following two constraints

$$
\begin{array}{rcl}
T^6x^6_2 + W^3x^6_3 &=& h^6 \\
T^7x^7_2 + W^3x^7_3 &=& h^7
\end{array}
$$

each with an independent set of variables. In case of example in Figure 1.2, for node $a(l_t) = 3$ we would have to add a constraint

$$x^6_2 = x^7_2.$$

Such a constraint is called a *nonanticipativity* or a *locking* constraint.

The complete set of nonanticipativity constraints for problem (1.4) may thus have the following form

$$\begin{array}{rcl} x_1^4 & = & x_1^5 \\ x_1^4 & = & x_1^6 \\ x_1^4 & = & x_1^7 \\ x_2^4 & = & x_2^5 \\ x_2^6 & = & x_2^7. \end{array} \tag{1.5}$$

There are other ways of representing the nonanticipativity constraints (the cyclical form is also frequently used).

## 1.4 Stochastic Programs in Algebraic Modeling Languages

The presence of two different sets associated with time and uncertainty dimensions in stochastic programs creates a difficulty to an algebraic modeling language. The uncertainty (or scenarios) needs to be indexed over time, and algebraic modeling languages normally do not provide such a facility. Consequently, none of the formulations of stochastic programs presented in Section 1.3 can be easily modeled in AMLs.

In this section, we make the assumption that probability distributions are discrete and that problems contain multiple stages or periods. Consequently, the problem can be represented in form of an event tree. This event tree is made of scenarios. It is quite usual to relate variables of a given node with those that correspond to the ancestor node in the previous stage. For example, any constraint that describes dynamics of the system would have such a form. However, the constraints which establish the link between the parent-child pair of nodes are particularly difficult to generate from the algebraic modeling language. The difficulty originates from the lack of standard description of the event tree or, more precisely, the lack of a tree-structured indexing system in AMLs.

When an AML generates the model, it performs extensive searches throughout the event tree. Therefore the way the event tree is described becomes crucial. Trees are obviously used in many computer science applications. There exist many different ways of describing and coding trees, and event trees used in stochastic programming could take advantage of these developments. Unfortunately, such techniques are not usually available from AMLs. The difficulty lies in the type of indexing system required to describe an event tree.

Trees like the one presented in Figure 1.2 are symmetric (every node except the leaves has the same number of children). Tricks exist such as the one used by Fragnière et al. (2000) to exploit the contiguity property to represent the symmetric tree and to retrieve easily the ancestor or the children of a given node (cf. Ahuja et al. 1993, pp. 774-776 for details). The idea is to use the breadth-first ordering of nodes in the event tree. Consider, for example, a regular (symmetric) tree with $d$ children at every node. Then the predecessor of node $i$ is the node $a(i) = \lceil \frac{i-1}{d} \rceil$, where the ceiling function $\lceil . \rceil$ rounds up the argument to the next integer. The

successors of node $i$ are nodes $id-d+2, id-d+3, \cdots, id+1$. Unfortunately, this addressing scheme cannot be generalized to unsymmetric event trees.

In many stochastic problems the discrete approximations of continuous distributions of random variables have various densities in different branches of the tree. Moreover, many models use trees that are automatically generated approximations of the stochastic process. These factors may lead to choosing highly unsymmetric event trees. Hence the restriction that only symmetric trees are modeled is unacceptable. The lack of efficient tree-structured indexing in algebraic formulations remains the main difficulty when AMLs are applied to generate stochastic programs. Although this could certainly be overcome at the cost of embedding some cumbersome generation schemes in AMLs, the major developers of AMLs hesitate before coding a devoted syntax to deal with stochastic programs in their modeling tools.

Modeling stochastic programs through AMLs is still in an early phase but several attempts have been made to standardize this process. The following brief literature review gives a nonexhaustive list of attempts made in this direction.

Gassmann and Ireland (1995, 1996) note that stochastic programming type modeling could greatly benefit from the implicit declaration of scenarios, via the declaration of random parameters. Buchanan et al. (2001) propose extensions to AML that allow recursive definition of stochastic dynamic problems and facilitate the link with sampling techniques. Leuba and Morton (1996) produce a complete SMPS format, i.e., the *core*, *time* and *stoch* files (cf. the article by Gassmann in this volume) directly from GAMS. Condevaux-Lanloy et al. (2001) extend the structure exploiting tool (Fragnière et al. 2000) to permit the formulation of the SMPS format from the algebraic modeling language. In their approach the time-related information is retrieved from the core model handled by the AML and the uncertainty information is loaded directly into the specialized SMPS based solver outside the AML. Entriken (2001) uses object-oriented programming techniques within the optimization modeling language to facilitate the management of stochastic programming models.

However, and in a general manner, we note the lack of standardization of modeling stochastic programs in AMLs. This has at least two reasons. Firstly, there is not yet a widely accepted syntax for a description of stochastic programs. Secondly, there is not yet a compact and flexible format in which AMLs could send the stochastic program to the specialized solver.

Below we illustrate the difficulties of tree-structured indexing in more detail when we use locking constraints to extend the deterministic inventory management problem to take uncertainty into account. The locking constraints have to be indexed over the event tree. This is done by hand in the model discussed. Next, we present the proposition made by the AMPL developers to model event trees.

### 1.4.1 Stochastic Extension of Multiperiod Inventory Model

In this section we present an extension of the multiperiod inventory problem that takes uncertainty into account. We use explicit locking constraints in the GAMS model presented in Section 1.2. Such an approach can be used to model both symmetric and unsymmetric event trees. However, it is rather tedious to implement.

Consider again the inventory problem (1.1). Suppose we introduce uncertainties in the future values of the demand parameters as represented by the event tree of Figure 1.4 and we model the problem as a stochastic program with recourse. This means that some decisions (activity levels) will be made after the information about the true value has been obtained. However, some decisions have to be taken immediately. These immediate decisions should take into account the expected cost of the recourse.

The stochastic model written in GAMS includes a new index `S` which stands for *scenarios*. Now the inventory balance constraint is generated for both the time and the scenario dimensions. As we have four scenarios, `h, l, m` and `a`, GAMS would generate four independent problems, each of them associated with a different set of data as indicated below.

```
TABLE
D(T,S)          market demand
              h    l    m    a
        0     0    0    0    0
        1     0    0    0    0
        2   200  200  150  150
        3   300  250  250  200
        4   400  400  400  400
        5     0    0    0    0
```

Let us observe that all generic constraints and variables include the index `S`. The objective in the stochastic inventory management problem is the expected value of the profit over all possible scenarios:

```
OBJECTIVE.. PROFIT=E=SUM((INDEX,S),PROB(S)*((P(INDEX)-2.0)*XMINUS(INDEX,S)
           -(P(INDEX)+2)*XPLUS(INDEX,S)-H*I(INDEX-1,S)));
INVBAL(T-1,S).. XMINUS(T,S)-XPLUS(T,S)+I(T,S)-I(T-1,S) =E= -D(T,S);
```

As has already been mentioned in Section 1.1, a possible way of dealing with uncertainty is to define and analyse several independent scenarios. The model is then solved independently for each scenario and the optimal solutions are gathered and compared with each other. This approach does not provide the unique first stage solution. Instead, it provides answers to "What-if" questions. The approach we shall present below extends it to ensure that the first stage decisions are identical for all scenarios.

We can achieve this by explicitly forcing the first stage decisions in all four otherwise independent scenarios to be identical. In the GAMS model we add three constraints `AFIOSS1`, `AFIOSS2` and `AFIOSS3` that fix the initial inventory to be identical in all four scenarios `h, l, m` and `a`. For example, the line

```
AFIOSS1..       I("0","h")=E=I("0","l");
```

forces $I_0$ in scenarios `h` and `l` to be the same, and the line

```
AFXMOSS1..      XMINUS("0","h")=E=XMINUS("0","l");
```

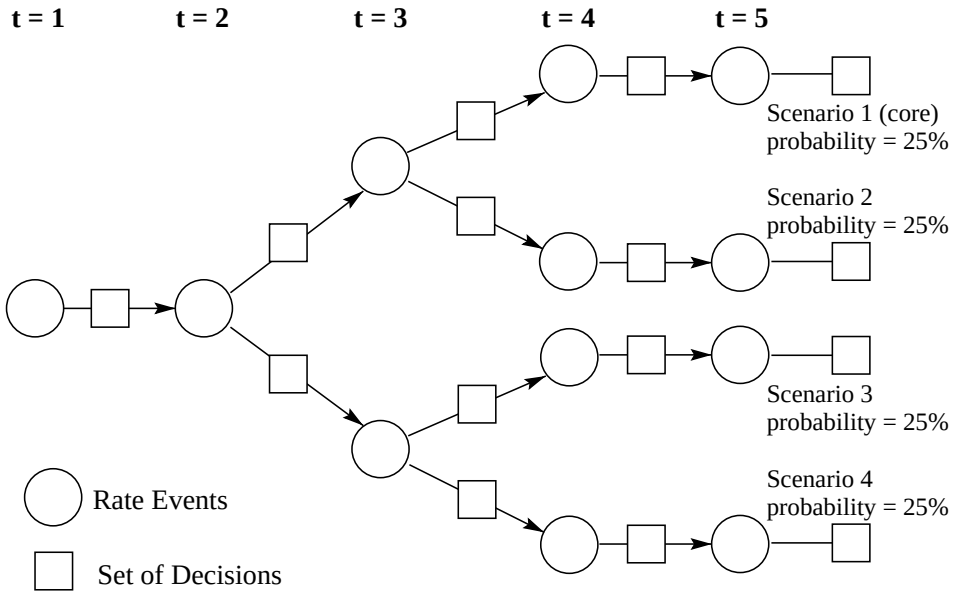



**Figure 1.4.** *Stochastic Invendeman model.*

forces $x_0^-$ in scenarios `h` and `l` to be identical. Several similar constraints are added for the variables at stage 1. At stage 2, however, only pairs of scenarios `(h, l)` and `(m, a)` are linked together. The number of necessary locking constraints is smaller than those in stages 0 and 1. All the constraints called `AF*` in the GAMS model are nonanticipativity constraints, where the last digit indicates the period in which the constraint is to be found.

The approach presented here requires explicit formulation of all nonanticipativity constraints and significantly increases the number of constraints in the model, as can be seen in the complete GAMS model given in Appendix .2. An important advantage of this approach is that it can be used for both symmetric and unsymmetric event trees. However, the approach is inefficient and prone to errors if a large number of nonanticipativity constraints has to be added. Using logical operators in set indexing would have allowed writing fewer locking constraints and leaving their generation to the algebraic modeling language.

The extensive formulation presented in this section illustrates clearly that the size of the simple model dramatically increases when the problem is transformed into a stochastic program. This type of stochastic programming formulation is therefore not tractable for large scale problems.

### 1.4.2 The AMPL Proposal

The developers of AMPL have proposed extensions to the syntax of their modeling language to allow a description of event trees. We reproduce their proposal below, following Fourer and Gay (1997). The modeling has been split into two steps. The first step consists in the definition of scenarios.

- `scenario` *scen-name*; Create a new *current* scenario. Inherit all set and parameter data from *parent* that was previously current. Incorporate subsequent data changes in *child* scenario only.
- `scenario` *scen-name* { `indexing` }; Create an indexed collection of scenarios.
- `scenario` *scen-name* `weight` *expr*; Associate a probability or other weight. *expr* denotes any arithmetic expression in current sets and parameters scenarios.

The second step adds scenarios referencing.

- `scenario` *scen-ref*; Make the indicated `scenario` current. *scen-ref* denotes either single *scen-name* or indexed *scen-name[object-ref]*.
- `scenario` *scen-name* `parent` *scen-ref*; Create new scenario having indicated parent, overriding the default (implicitly build a tree of scenarios).
- `nscens, scenname[`*expr*`], scen[`*expr*`]`. Extension of AMPLs generic names to scenario references (loop over all scenarios in the tree).

This approach has not been implemented yet. When implemented it would allow building stochastic programming models of small to medium sizes. However, the proposal does not give a clue on how a large unsymmetric event tree can be modeled within AMPL.

More generally, the developers of algebraic modeling languages do not want to commit their languages to a specific syntax of event tree description. This syntax is closely related to a standard in which problems are described in the AMLs and the format in which they are passed to the specialized stochastic programming solvers.

## 1.5 SP Solution Techniques Available to AMLs

At the moment of writing this chapter, the only option available in AMLs is to generate the full deterministic equivalent. The only alternative left is thus to use the general purpose solvers that by default would use a direct solution method to tackle the problem. This approach is quite efficient as long as the problem is small to medium size and can be generated within memory limits. The need for accurate modeling of stochastic processes inevitably leads to a size explosion in the model. Even if the user is satisfied with the accuracy of the generated problem, and the general purpose solver can solve this problem efficiently, there is a danger that the generation of the problem significantly contributes to the overall solution process. It is not unusual, for example, that model generation by an AML takes more time than the following solution of the problem. Gondzio and Kouwenberg (2001) have generated a medium scale stochastic model by the GAMS modeling language and a specialized generation program (Kouwenberg 1999). The latter was 815 times faster.

Over the years many specialized techniques have been developed for stochastic programming. They usually exploit special structure of the problem. Many

of these techniques rely on some variant of Benders decomposition (Van Slyke and Wets 1969). The decomposition approach breaks the very large problem into smaller manageable optimization problems. This has several advantages. First, the peak memory requirement (needed to generate and then to read the deterministic equivalent problem) can be avoided. Additionally, the problem can be passed to the solver in pieces that are suitable for the decomposition approach. Therefore, as has been observed by Fragnière et al. (2000), within the same memory limits decomposition-based solvers can deal with problems that are at least an order of magnitude larger than those solvable by a direct approach.

An alternative would consist in implementing simple decomposition technique directly within AMLs. This approach is routinely used in certain economical applications: the decomposition loops are programmed in GAMS, for example, in the context of nonlinear stochastic programming problems (Chang and Fragnière 1996). Indeed, the presence of procedural statements such as `if-then-else` and `do-while` provided by most AMLs makes it possible to implement simple optimization algorithms. The interested reader can consult the library of examples of algorithms implemented through AMPL which includes Benders decomposition (Fourer and Gay 1999). The article by Gassmann and Gay in this volume shows how to implement a nested Benders algorithm within the AMPL control language. The authors conclude that such an approach cannot be generalized because the AML-based decomposition algorithm depends upon the syntax used by a particular model and is not reusable in a different model. Moreover, the AML is not necessarily the best environment to implement complicated optimization algorithms needed to solve stochastic programming problems efficiently.

Although several efficient algorithms have been proposed for stochastic programming, the limitations discussed so far prevent access to many of these techniques from AMLs. Indeed, the research in stochastic programming provides evidence that very large problems can be generated and solved. The research results on solution techniques are very much ahead of current links to solvers available in AMLs.

Below we recall some of the results that indicate currently achievable limits of solvable problems. We underline that all the solution techniques use parallel computing. Yang and Zenios (1997) solved test problems with up to 2.6 million constraints and 18.2 million variables. They used a parallel direct interior point method. Gondzio and Kouwenberg (2001) solved a financial planning problem with 7 decision stages and a total of 5 million scenarios at the planning horizon, the linear program consisting of 12.5 million constraints and 25 million variables. They used an interior point based variant of Benders decomposition run on a 16-processor parallel machine. Blomvall and Lindberg (2000) solved a problem with 10 stages and 1.9 million scenarios, resulting in a separable convex program with 119 million constraints and 67 million variables. They used a direct interior point method with a specialized Riccati-based solver for computing Newton directions and ran it on a Beowulf cluster of 32 PCs. Linderoth and Wright (2001) solved a problem with 10 million scenarios, the linear program having 985 million constraints and 12,600 million variables (see also the article by Linderoth and Wright in this volume). They used a variant of Benders decomposition and ran it on a grid of 1345 workstations.

To conclude, there is a need to improve the links between the AMLs and the solvers. Attempts have already been made that go into this direction. For example, Fragnière et al. (2000) have used GAMS to generate a one million scenario problem, a linear program with 1111112 constraints and 2555556 variables. The problem was solved by a specialized parallel interior point based decomposition algorithm running on a cluster of 10 Linux PCs. The solver was accessed directly from the AML. Still the problem was passed to the solver in a deterministic equivalent form. This approach clearly demonstrates the need for improving the link between the AML and the specialized solver to avoid the bottleneck generation of the deterministic equivalent. We address this issue in the next section.

## 1.6 Communication Between Solver and AML

Every AML has a set of specialized routines to communicate with the solver. Usually, the whole problem is passed at once to the solver in form of a text or binary file. This implies that sufficient memory has to be available to store the complete mathematical program. Typically AMLs generate the stochastic programming problem in the deterministic equivalent form and call a general-purpose optimization code to solve it. The size of the deterministic equivalent problem is proportional to the number of nodes in the event tree. Therefore, the AML may require a vast amount of memory to store it. As has already been mentioned, the real bottleneck is often not the memory requirement but the time of the problem generation.

At least some of the earlier mentioned drawbacks of the problem generation by AMLs could be avoided if the SMPS format were used. Moreover, any efficient solution method for stochastic programming is built upon the exploitation of the special structure of the problem, and the complete structure information is available from the SMPS format. At the moment of writing this chapter AMLs cannot generate stochastic problems in SMPS format. However, several attempts to overcome this difficulty have already been made (Buchanan et al. 2001; Condevaux-Lanloy et al. 2001; Messina and Mitra 1997). The problem has been treated in different ways. One of them consists in developing the extensions of existing AMLs dedicated to stochastic optimization.

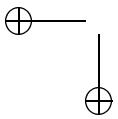
Although s-Magic (Buchanan et al. 2001) does not produce SMPS format from the problem, it uses the recursive definition and communicates with the solver using a specialized memory efficient description of the problem. The problem is represented in a compact format close in spirit to SMPS. Stochastic extension (Condevaux-Lanloy et al. 2001) of the structure exploiting tool (Fragnière et al. 2000) uses the AML to generate the deterministic part of the model in form of the *core* and *time* files in the SMPS format. The information of uncertainty is produced outside the AML and communicated directly to the specialized solver. SPInE (cf. Messina and Mitra 1997 and the article by Valente et al. in this volume) is a closed modeling system that generates the SMPS format of the stochastic programming problem and has access to built-in specialized optimization tools — the Benders decomposition. Direct solution of the deterministic equivalent form of the problem is also available as an option.

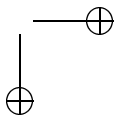
## 1.7 Conclusions

Stochastic programming is a promising technology for handling planning problems in uncertain environments. At least this has always been said since the publication of the seminal paper on linear programming under uncertainty (Dantzig 1955). Unfortunately, due to modeling difficulties this technology has not yet reached the wide audience it deserves. To facilitate incorporating uncertainty in the planning models, user-friendly modeling systems are needed that can access the stochastic programming technology. Widely used algebraic modeling languages are candidates to close this gap.

In this chapter we underlined the difficulties in the use of uncertainty in the modeling of real-life problems. We began our exposé with a discussion of an inventory problem. Deterministic formulations of this problem in the algebraic modeling language and in mathematical terms are very similar to each other. Then we explained the modification to include a stochastic dimension (uncertain demands) into the problem. Although the problem remains simple, it illustrates all the difficulties of including stochastic programs into modeling systems. We elaborated on different approaches that allow writing stochastic programs directly in algebraic modeling languages. We ended the chapter with a discussion of stochastic programming solution techniques accessible from modeling systems. These systems certainly need further development to reach industry standard. We expect that this progress will be made in the next few years and the integrated modeling system for stochastic programming will enable the modelers to popularize the stochastic programming technology through relevant applications.

**Acknowledgement**
We are grateful to Gus Gassmann for constructive comments, resulting in an improved presentation. The research of the first author was supported by the Fonds National de la Recherche Scientifique Suisse, grant #1213-058892.99/1. The research of the second author was supported by the Engineering and Physical Sciences Research Council of UK, EPSRC grant GR/M68169.





# Bibliography


Ahuja, R. K., T. L. Magnanti, and J. B. Orlin (1993). *Network Flows*. New York: Prentice-Hall.

Birge, J., M. Dempster, H. Gassmann, E. Gunn, A. King, and S. Wallace (1987). A standard input format for multiperiod stochastic linear programs. *Committee on Algorithms Newsletter 17*, 1–19.

Bisschop, J. and R. Entriken (1993). AIMMS: The modeling system. Paragon Decision Technology.

Blomvall, J. and P. O. Lindberg (2000). A Riccati-based primal interior point solver for multistage stochastic programming - extensions. Technical report, Department of Mathematics, Linköping University, 58183 Linköping. To appear in: *Optimization Methods and Software*.

Brooke, A., D. Kendrick, and A. Meeraus (1992). *GAMS: A User's Guide*. Redwood City, California: The Scientific Press.

Buchanan, C., K. McKinnon, and G. Skondras (2001). The recursive definition of stochastic linear programming problems within an algebraic modeling language. *Annals of Operations Research 104*(1/4), 15–32.

Chang, D. and E. Fragnière (1996). SPLITDAT and DECOMP: Two new GAMS I/O subroutines to handle mathematical programming problems with an automated decomposition procedure. Stanford University, Department of Operations Research, manuscript.

Condevaux-Lanloy, C., E. Fragnière, and A. J. King (2001). SISP: a simplified interface for stochastic program. To appear in: *Optimization Methods and Software*.

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

Entriken, R. (2001). Language constructs for modeling stochastic linear programs. *Annals of Operations Research 104*(1/4), 49–66.

Fourer, R. and D. Gay (1997). Proposals for stochastic programming in the AMPL modeling language. International Symposium on Mathematical Programming, Lausanne.

Fourer, R. and D. Gay (1999). Implementing algorithms through AMPL scripts (looping and testing 2). AMPL Web Page, URL: `http://www.ampl.com/cm/cs/what/ampl/NEW/LOOP2/index.html`.

Fourer, R., D. Gay, and B. W. Kernighan (1993). *AMPL: A Modeling Language for Mathematical Programming*. San Francisco, California: The Scientific Press.

Fragnière, E. and J. Gondzio (2002). Optimization modeling languages. In P. Pardalos and M. Resende (Eds.), *Handbook of Applied Optimization*, pp. 993–1007. New York: Oxford University Press.

Fragnière, E., J. Gondzio, R. Sarkissian, and J.-P. Vial (2000). Structure exploiting tool in algebraic modeling languages. *Management Science 46*(8), 1145–1158.

Fragnière, E., J. Gondzio, and J.-P. Vial (2000). Building and solving large-scale stochastic programs on an affordable distributed computing system. *Annals of Operations Research 99*(1/4), 167–187.

Gassmann, H. and A. Ireland (1995). Scenario formulation in an algebraic modelling language. *Annals of Operations Research 59*, 45–75.

Gassmann, H. and A. Ireland (1996). On the automatic formulation of stochastic linear programs. *Annals of Operations Research 64*, 83–112.

Gondzio, J. and R. Kouwenberg (2001). High performance computing for asset liability management. *Operations Research 49*(6), 879–891.

Kouwenberg, R. (1999). *LEQGEN: A C-tool for generating linear and quadratic programs, User's Manual*. Rotterdam, The Netherlands: Econometric Institute, Erasmus University.

Leuba, A. and D. Morton (1996). Generating stochastic linear programs in S-MPS format with GAMS. INFORMS Conference, Atlanta.

Linderoth, J. and S. J. Wright (2001). Decomposition algorithms for stochastic programming on a computational grid. Technical Report MCS-P875-0401, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA.

Messina, E. and G. Mitra (1997). Modelling and analysis of multistage stochastic programming problems: A software environment. *European Journal of Operational Research 101*, 343–359.

Thompson, G. (1992). *Computational Economics*. New York: Scientific Press.

Van Slyke, R. and R. J.-B. Wets (1969). L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal of Applied Mathematics 17*, 638–663.

Yang, D. and S. A. Zenios (1997). A scalable parallel interior point algorithm for stochastic linear programming and robust optimization. *Computational Optimization and Applications 7*, 143–158.

# .1 The GAMS Model of Deterministic Inventory Problem

```
SETS
  T    time periods  /0,1,2,3,4,5/
  INDEX(T)           /  1,2,3,4,5/
  OPENING(T)         /0          /
  TERMINAL(T)        /          5/;
PARAMETERS
  P(INDEX)   market price
               /1   75
                2   65
                3   89
                4   77
                5   80/
  D(T)       market demand
               /0    0
                1    0
                2  200
                3  300
                4  400
                5    0/;
VARIABLES
  PROFIT
POSITIVE VARIABLES
  XMINUS(T)   quantity sold at time T
  XPLUS(T)    quantity bought at time T
  I(T)        inventory held at time T;
EQUATIONS
  OBJECTIVE   calculating net profit
  INVBAL(T)   inventory balance at time T;
OBJECTIVE..   PROFIT =E= SUM(INDEX,(P(INDEX)-2.0)*XMINUS(INDEX)
              -(P(INDEX)+2.0)*XPLUS(INDEX)-I(INDEX-1));
INVBAL(T-1).. XMINUS(T)-XPLUS(T)+I(T)-I(T-1)  =E=  -D(T);
I.UP(T)        = 500;
I.FX(OPENING)  = 300;
I.FX(TERMINAL) = 300;
MODEL INVENDEMAN/ALL/;
SOLVE INVENDEMAN USING LP MAXIMIZING PROFIT;
```

## .2 The GAMS Model of Stochastic Inventory Problem

```
SETS
  S    scenarios     /h,l,m,a/
  T    time periods  /0,1,2,3,4,5/
  INDEX(T)           /  1,2,3,4,5/
  OPENING(T)         /0          /
  TERMINAL(T)        /          5/;
PARAMETERS
  P(INDEX)   market price
              /1   75
               2   65
               3   89
               4   77
               5   80/
  PROB(S)    scenario probabilities
              /h   0.25
               l   0.25
               m   0.25
               a   0.25/
TABLE
  D(T,S)     market demand
                     h    l    m    a
                0    0    0    0    0
                1    0    0    0    0
                2  200  200  150  150
                3  300  250  250  200
                4  400  400  400  400
                5    0    0    0    0
VARIABLES
  PROFIT
POSITIVE VARIABLES
  XMINUS(T,S)   quantity sold at time T
  XPLUS(T,S)    quantity bought at time T
  I(T,S)        inventory held at time T;
EQUATIONS
  OBJECTIVE     calculating net profit
  INVBAL(T,S)   inventory balance at time T
  AFIOSS1
  AFIOSS2
  AFIOSS3
  AFXMOSS1
  AFXMOSS2
  AFXMOSS3
  AFXPOSS1
  AFXPOSS2
  AFXPOSS3
  AFI1SS1
  AFI1SS2
```

```
  AFI1SS3
  AFXM1SS1
  AFXM1SS2
  AFXM1SS3
  AFXP1SS1
  AFXP1SS2
  AFXP1SS3
  AFI2SS1
  AFI2SS2
  AFXM2SS1
  AFXM2SS2
  AFXP2SS1
  AFXP2SS2 ;
OBJECTIVE..   PROFIT =E= SUM((INDEX,S),PROB(S)*((P(INDEX)-2.0)*XMINUS(INDEX,S)
            -(P(INDEX)+2.0)*XPLUS(INDEX,S)-H*I(INDEX-1,S)));
INVBAL(T-1,S).. XMINUS(T,S)-XPLUS(T,S)+I(T,S)-I(T-1,S) =E= -D(T,S);
AFI0SS1..          I("0","h") =E= I("0","l");
AFI0SS2..          I("0","m") =E= I("0","a");
AFI0SS3..          I("0","m") =E= I("0","l");
AFXM0SS1..         XMINUS("0","h") =E= XMINUS("0","l");
AFXM0SS2..         XMINUS("0","m") =E= XMINUS("0","a");
AFXM0SS3..         XMINUS("0","m") =E= XMINUS("0","l");
AFXP0SS1..         XPLUS("0","h")  =E= XPLUS("0","l");
AFXP0SS2..         XPLUS("0","m")  =E= XPLUS("0","a");
AFXP0SS3..         XPLUS("0","m")  =E= XPLUS("0","l");
AFI1SS1..          I("1","h") =E= I("1","l");
AFI1SS2..          I("1","m") =E= I("1","a");
AFI1SS3..          I("1","m") =E= I("1","l");
AFXM1SS1..         XMINUS("1","h") =E= XMINUS("1","l");
AFXM1SS2..         XMINUS("1","m") =E= XMINUS("1","a");
AFXM1SS3..         XMINUS("1","m") =E= XMINUS("1","l");
AFXP1SS1..         XPLUS("1","h")  =E= XPLUS("1","l");
AFXP1SS2..         XPLUS("1","m")  =E= XPLUS("1","a");
AFXP1SS3..         XPLUS("1","m")  =E= XPLUS("1","l");
AFI2SS1..          I("2","h") =E= I("2","l");
AFI2SS2..          I("2","m") =E= I("2","a");
AFXM2SS1..         XMINUS("2","h") =E= XMINUS("2","l");
AFXM2SS2..         XMINUS("2","m") =E= XMINUS("2","a");
AFXP2SS1..         XPLUS("2","h")  =E= XPLUS("2","l");
AFXP2SS2..         XPLUS("2","m")  =E= XPLUS("2","a");
I.UP(T,S)        = 500;
I.FX(OPENING,S)  = 300;
I.FX(TERMINAL,S) = 300;
MODEL INVENDEMAN/ALL/;
SOLVE INVENDEMAN USING LP MAXIMIZING PROFIT;
```