Extending algebraic modelling languages for Stochastic Programming

Christian Valente, Gautam Mitra, Mustapha Sadki

CARISMA, Brunel University and OptiRisk Systems

Uxbridge, Middlesex, UK

Robert Fourer

Northwestern University, Evanston, IL

Abstract

Algebraic modelling languages (AML) have gained wide acceptance and use in Mathematical Programming by researchers and practitioners. At a basic level, stochastic programming models can be defined using these languages by constructing their deterministic equivalent. Unfortunately, this leads to very large model data instances. We propose a direct approach in which the random values of the model coefficients and the stage structure of the decision variables and constraints are "overlaid" on the underlying deterministic (core) model of the SP problems. This leads not only to a natural definition of the SP model, the resulting generated instance is also a compact representation of the otherwise large problem data. The proposed constructs enable the formulation of two stage and multistage scenario based recourse problems. The design is presented as a stochastic extension of the AMPL language which we call SAMPL; this in turn is embedded in an environment called SPInE (Stochastic Programming Integrated Environment) which facilitates modelling and investigation of SP problems.

Table of Contents

1 Introduction

1.1 Background

Over the past decades, stochastic programming models have gained wide acceptance by practitioners in the field of optimisation and operations research. Research in financial areas such as portfolio management and pension funds management (Ziemba and Mulvey 1998), in manufacturing networks and supply chain optimisation (Eppen, Martin et al. 1989; Lucas, Messina et al. 1997; Escudero, Galindo et al. 1999; Shapiro 2001), as well as in energy and environmental planning (Pereira and Pinto 1985; Bosetti, Messina et al. 2002), has demonstrated that the solution of stochastic programming models is superior to that of deterministic models in terms of expectation over future events. Stochastic programming models explicitly consider uncertainty in some of the model parameters, and provide optimal decisions which are hedged against such uncertainty.

Unfortunately, the size of stochastic programming problems grows exponentially with the number of decision stages considered by the models and with the number of possible outcomes of the random parameters at each stage. This is a challenging issue both in terms of modelling and solution algorithms. The implementation of large scale mathematical programming models requires the generation of matrices which represent the problem constraints and that are stored in machine readable form in order to be interpreted by the solvers. This generation process used to be performed by ad-hoc software modules, and required considerable development effort. For deterministic models, the drawbacks of this approach were overcome with the advent of a class of declarative programming languages known as *Algebraic Modelling Languages* (AML), which are powerful and flexible tools for the generation of model data instances. The entities of the optimisation models (sets, decision variables, parameters, constraints, objectives etc.) are defined in a way that closely resembles the algebraic notation. AMPL (Fourer, Gay et al. 1993), GAMS (Brooke, Kendrick et al. 1998), AIMMS (Bisschop and Roelofs 1999) and MPL (Maximal Software 2002) are amongst the most commonly used AMLs. Most of these support the formulation of Linear Programs (LP), Mixed Integer Programs (MIP), Quadratic Programs (QP) and recently Nonlinear Programs (NLP), and are capable of generating model instances using machine readable formats.

Stochastic programming models are characterised by the presence of uncertainty associated with some of the model parameters. It is appropriate to consider these to be random variables with a specified probability distribution. Unfortunately, the existing modelling languages do not provide specific constructs for the definition of such random model parameters. In this paper we present a direct approach for the definition of SP models via declarative syntax, and illustrate how such a concept can be applied to extend virtually any algebraic modelling language. We consider the AMPL systems and by means of examples, define the syntax of the stochastic extension SAMPL.

1.2 A case for SP extensions to AMLs

Optimum decision problems under uncertainty may be posed as (a) two stage or (b) multistage scenario based recourse problem or (c) a chance constrained programming problem. For a complete discussion of these classes of models the readers are referred to (Birge and Louveaux, 1997), (CARISMA Lecture Notes, 2006) and (SPInE Manual, 2004). The classes of models (a) and (b) above are essentially individual LPs, one for each scenario, connected together across the stages by constraints which link the decision variables. This leads to a highly structured LP model. Therefore in principle an AML can be used to generate such SP models. We as well as a number of researchers in this field (King, A.J. 1994), (Kall and Mayer 2005) (Fourer and Lopes, 2004) (Gassmann, 1996) have identified a few compelling arguments for designing extensions to AMLs which facilitates SP model formulation and investigation.

Stochastic Programming opens up the possibility of a rich modelling paradigm connecting decision modelling with the descriptive simulation modelling, see DiDomenica, et al (2006,2007).

In general SP analysis leads to a complete investigation of a family of models (expected value problems, scenario problems, Wait and See, Here and Now, see section 2.4) and some related stochastic information such as expected value of perfect information (EVPI), and value of stochastic solution (VSS). Hence extending AMLs with language constructs which makes it easier to generate and investigate such a family of models is a clear advantage.

Also, a major progress in computational stochastic programming took place in late eighties with the publication of SMPS model representation standard. The use of SMPS standard format has encouraged SP modellers to generate models in such external forms; it has also facilitated the communication to a solver in a suitable internal form. It's well known that even the most efficient solvers do not scale up effectively with growth in SP model size (due to scenarios), therefore modelling system and solver coupling in a compact form is essential. The scenarios presented as a scenario tree of parameter values not only provides the coupling between data and decision model, it also determine the staircase or hierarchical structures for the two stage and multistage SPs. Hence a modelling system which provides this two fold coupling of scenarios to data and model structure is clearly a desirable formulation tool for SP modelling. In the different sections of this paper whenever we consider it appropriate we highlight the modelling advantages gained by introducing these language constructs for SP within an AML.

The rest of the paper is organised as follows:

In section 2, we define some classes of models for optimum decision making under uncertainty. In section 3, we introduce an illustrative model of a two stage and multistage stochastic programming problem with recourse. In section 4, we analyse the modelling requirements for stochastic programming problems and introduce our approach in extending algebraic modelling languages. We define new language constructs for AMPL and use the resulting SAMPL extension to formulate the illustrative model. In section 5 we consider the practical implementation of these stochastic programming extensions; this will include a brief presentation of SPInE, the system we implemented that is able to parse and solve SAMPL models. We conclude in section 6 with some final remarks and future research directions.

2 Modelling stochastic programming problems

2.1 Classes of SP problems

A full taxonomy of SP problems shown in Figure 1 is discussed in (Wallace and Ziemba, 2005); this book also contains examples of many applications of SP.
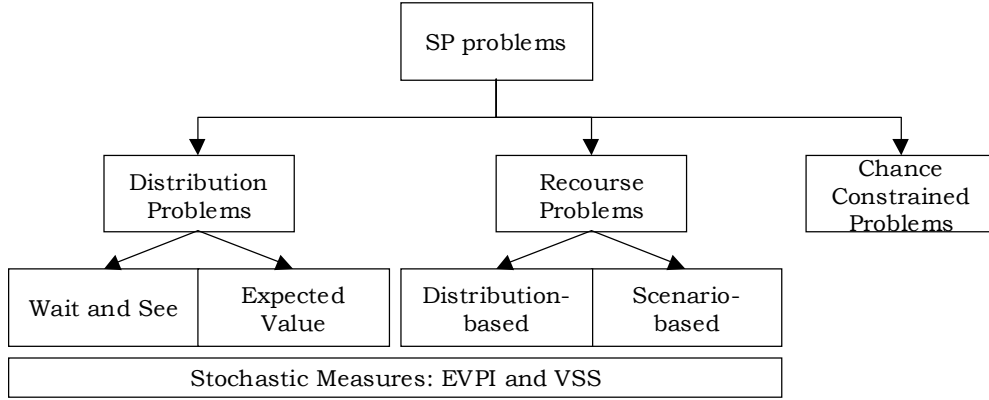
Figure 1 Taxonomy of SP problems.

Our extensions to AMLs address only a subclass of these SP problems, namely, two stage and multistage recourse problems. We first introduce these problems and then define the related stochastic information.

Consider the linear programming problem:

$$Z = \min cx,$$
$$subject\ to \quad Ax = b,$$
$$x \geq 0,$$
$$where \quad A \in R^{m \times n}; c, x \in R^{n}; b \in R^{m}.$$

(1)

Let (Ω, F, P) denote a (discrete) probability space where $\omega \in \Omega$ denote the events. Let us denote the realizations of *A, b, c* for a given ω as:

$$\left(A^{\omega}, b^{\omega}, c^{\omega}\right) = \xi^{\omega} \quad or \quad \xi(\omega).$$

(2)

We use $p^{\omega} = p(\xi(\omega))$ to denote the probability associated with the realisation $\xi(\omega)$. The classes of stochastic models illustrated in Figure 1 are defined in the following sections.

The Expected Value Problem

The Expected Value (EV) model is constructed by replacing the random parameters by their expected values. Such EV model is thus a linear program, as the uncertainty is dealt with before it is introduced into the underlying linear optimisation model. It is common practice to formulate and solve the EV problem in order to gain some insight into the decision problem. Let the constraint sets corresponding to the problem stated in (1) and (2) be defined as:

$$F^{\omega} = \left\{x \mid A^{\omega} x = b^{\omega}, x \geq 0\right\} \quad for \quad \left(A^{\omega}, b^{\omega}, c^{\omega}\right) \ or \ \xi(\omega).$$

(3)

We can reconsider (1) as an expected value or an average value problem where:

$$(\overline{A}, \overline{b}, \overline{c}) = \overline{\xi} = E[\xi^\omega] = \sum_{\omega \in \Omega} p^\omega \xi^\omega,$$

$$Z_{EV} = \min \overline{c}x, \tag{4}$$

subject to

$$\overline{A}x = \overline{b}.$$

Let x_{ev}^* denote the optimal solution to the above problem. This solution can be evaluated for all possible realisations $\omega \in \Omega$. We can thus determine the corresponding objective function values and compute what is called the expectation of the expected value solution:

$$Z_{EEV} = E[c^\omega x_{ev}^*]. \tag{5}$$

If, however, an ω exists such that: $x_{ev}^* \notin F^\omega$, i.e. x_{ev}^* is not feasible for some realisations of the random parameters, we set:

$$Z_{EEV} \to +\infty. \tag{6}$$

Wait and See Problems

Wait and See (WS) problems assume that the decision-maker is somehow able to wait until the uncertainty is resolved before implementing the optimal decisions. This approach therefore relies upon perfect information about the future. Because of its very assumptions such a solution cannot be implemented and is known as the "passive approach". Wait and see models are often used to analyse the probability distribution of the optimum objective value, and consist of a family of LP models, each associated with an individual scenario in the event tree. The corresponding problem is stated as:

$$Z^\omega = \min c^\omega x,$$
$$A^\omega x = b^\omega. \tag{7}$$

The expected value of the wait and see solutions is defined as:

$$Z_{ws} = E[Z^\omega] = \sum_{\omega \in \Omega} Z^\omega p^\omega. \tag{8}$$

2.2   Stochastic Programming Problems with Recourse

A simple (single stage) stochastic programming model can be formulated as follows:

$$Z_{HN} = \min E[c^\omega x],$$
$$\text{where} \quad x \in F \tag{9}$$

$$\text{and} \quad F = \bigcap_{\omega \in \Omega} F^\omega. \tag{10}$$

The term *Here and Now* (HN) is often used to refer to stochastic programming problems, reflecting the fact that decisions are taken before perfect information on the states of nature is revealed.

The formulation of the classical two-stage SP model with recourse is as follows:

$$Z_{HN} = \min \quad cx + E_\omega Q(x, \omega),$$
$$\text{subject to} \quad Ax = b, \tag{11}$$
$$x \geq 0,$$

where:

$$Q(x, \omega) = \min f(\omega) y(\omega),$$
$$\text{subject to} \quad D(\omega) y(\omega) = d(\omega) + B(\omega) x, \tag{12}$$
$$y(\omega) \geq 0,$$
$$\omega \in \Omega.$$

The matrix A and the vector b are known with certainty. The function $Q(x, \omega)$, referred to as the recourse function, is in turn defined by the linear program set out in (12). The technology matrix $D(\omega)$, also known as the recourse matrix, the right-hand side $d(\omega)$, the inter-stage linking matrix $B(\omega)$, and the objective function coefficients $f(\omega)$ of this linear program are random. For a given realisation ω, the corresponding recourse action $y(\omega)$ is obtained by solving the problem set out in (12).

The generalisation of the two-stage recourse problem is known as multistage stochastic programming problem with recourse, as the future unfolds in several sequential steps and subsequent recourse actions are taken. A decision made in stage t should take into account all future realisations of the random parameters and such decisions only affect the remaining decisions in stages t+1 … T. In stochastic programming this concept is known as non-anticipativity. The general formulation of a multistage recourse problem is set out in equations (13) - (15) below:

$$Z_{HN} = \min_{x_1} \left\{ c_1 x_1 + E_{\xi_2} \left[\min_{x_2} c_2 x_2 + E_{\xi_3|\xi_2} \left[\min_{x_3} c_3 x_3 + \ldots + E_{\xi_T|\xi_{T-1}|\ldots|\xi_2} \min_{x_T} c_T x_T \right] \right] \right\}, \tag{13}$$

subject to:

$$\begin{array}{lllll}
A_{11} x_1 & & & = b_1, \\
A_{21} x_1 + & A_{22} x_2 & & = b_2, \\
A_{31} x_1 + & A_{32} x_2 + & A_{33} x_3 & = b_3, \\
\vdots & & \ddots & \vdots \\
A_{T1} x_1 + & A_{T2} x_2 + & A_{T3} x_3 + & \ldots + A_{TT} x_T = b_T.
\end{array} \tag{14}$$

In (13) and (14) $t=1,\ldots,T$ represents the planning horizon and the vectors:

$$\xi_t = (b_t, c_t, A_{t1}, \ldots A_{tT}) \quad \forall t \in [2, \ldots, T] \tag{15}$$

are random variables on a probability space (Ω, F, P). It is important to stress the difference between decision stages and model time periods. Although these coincide in many applications, stage can be

regarded in general as a time period where new information about the state of nature is provided, that is the realisation of the random vectors can be observed.

2.3 Stochastic measures

An important aspect of formulating and processing SP problems is to collect stochastic information regarding the family of models and also to compute the related bounds, see (Birge and Louveaux, 1997) and (CARISMA Lecture Notes).

Value of stochastic solution (VSS) and the expected value of perfect information (EVPI) are two stochastic measures which are defined as:

$$\text{VSS} = Z_{EEV} - Z_{HN} \,, \tag{16}$$

$$\text{EVPI} = Z_{HN} - Z_{WS} \,. \tag{17}$$

3 An illustrative example

In this section, we introduce a distribution logistic application to illustrate two stage SP, multistage SP and the expected value problems. We first set out the problem, then using scenarios to define the random parameters we describe a two stage SP model followed by a multistage version of the same model. We use these example models to highlight the inadequacies of the current modelling languages in terms of SP modelling and we show how we use our SP language constructs to overcome these limitations.

3.1 A distribution logistic model

We state the distribution logistic problem as follows:

A clothing manufacturer produces goods in two factories. The manufacturer supplies the retailers, called dealers with three products, Shirts, Skirts and Jeans. The products are shipped to three main dealers in quantities of tens of thousands. The manufacturer can carry over inventory from one period to the next in the factories. A link or inventory variable is introduced which represents the amount of products transferred from one period to the next. The manufacturer knows the production costs, transportation costs, inventory costs for the next month with certainty. For simplicity it is assumed that all costs, production and inventory capacities are known and constant over the time horizon whereas the dealer requirements can vary for each time period (e.g. for seasonal reasons). Production rates and the initial inventory of each product at each factory are known as well. A shortage penalty is introduced if the demand is not met. Figure 2 shows the flow of products in the logistic network.
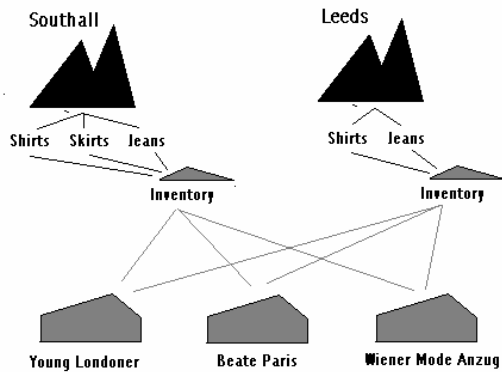
Figure 2. Physical network structure

For simplicity we assume that uncertainty only occurs in the demand for the products in each of the future time periods $t = 1, 2, 3, 4$. If we assume that we know the expected value of the demands then we can formulate a dynamic linear programming problem which is a deterministic EVLP model.

3.2   Introducing scenarios

In the problem stated above, we consider the demand as a random parameter characterized by known probability distribution. A simple specification of these scenarios is shown in Figure 3: given an initial value for the demand, we postulate a rule which takes into account two possible realisations of the demand in each time period $t>1$. The percentage at each node represents the change of the demand from the value at the parent node and the amount of the change is assumed the same for all the products and across all the dealers. The resulting scenario tree consists of a set of scenarios $\Xi=[1,2..,8]$. At each node, the probabilities for the two possible realisation are equal (*Pr=0.5*), thus the resulting scenarios are equiprobable with probability *Pr[s]=0.5*0.5*0.5=0.125*. The demand parameter is defined as:

d_{jkts} is the requirement of dealer k for product j at time period t under scenario s.
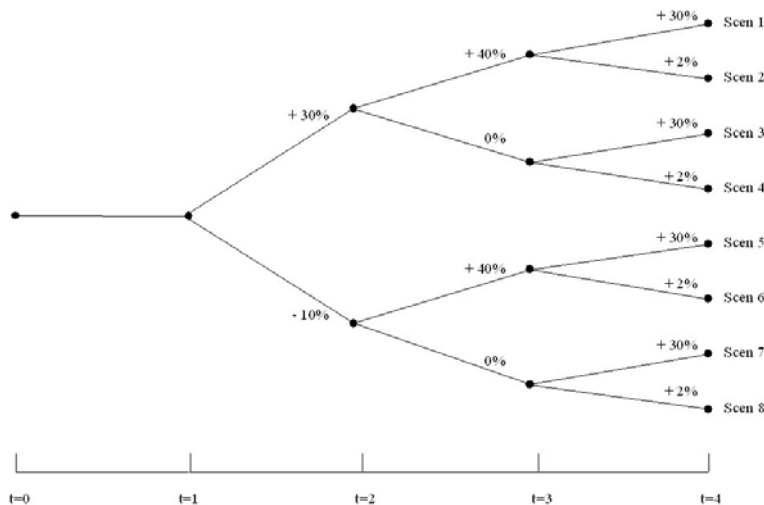

Figure 3. Scenario tree specification for the demand.

3.3 Stochastic Programming formulations

We now formulate the two stage stochastic programming problem with recourse using the sets, indices, parameter values and decision variables described below.

Indices and dimensions

$i = 1,\ldots,NF$ denotes the factories $(NF=2)$,

$j = 1,\ldots,NP$ denotes the products $(NP=3)$,

$k= 1,\ldots,NK$ denotes the dealers $(NK=3)$,

$t = 1,\ldots,NT$ denotes the time period $(NT=4)$,

$s= 1,\ldots,NS$ denotes the scenarios $(NS=8)$.

Parameters

q_{ji} the cost of producing one unit of product j at factory i,

c_{ik} the cost of transporting one unit of any product from factory i to dealer k,

v_{ji} the cost for holding one unit of product j at factory i on inventory,

p_{jk} the shortage penalty for product j at dealer k,

a_{ji} the production capacity of product j at factory i,

n_{ji} the inventory capacity of product j at factory i,

d_{jkts} the requirement of dealer k for product j at time period t under scenario s,

l_{ji} the initial inventory of product j at factory i,

π_s the probability for scenario s.

Decision variables

x_{jits} the number of units of product j manufactured at factory i at time t, scenario s,

z_{jikts} the number of units of product j sent from factory i to dealer k at time t, scenario s,

y_{jits} the number of units of product j held at inventory at factory i at time t, scenario s,

w_{jkts} the shortage of units of product j at dealer k at time t, scenario s.

The objective of the recourse problems is to minimise the the expectation of the total costs over all scenarios.

Objective function

$$Min \quad Costs = \sum_{s=1}^{NS} \pi_s \cdot \Big(\sum_{j=1}^{NP}\sum_{i=1}^{NF}\sum_{t=1}^{NT} q_{ji} \cdot x_{jits} + \sum_{j=1}^{NP}\sum_{i=1}^{NF}\sum_{k=1}^{NK}\sum_{t=1}^{NT} c_{id} \cdot z_{jikts}$$
$$+ \sum_{j=1}^{NP}\sum_{i=1}^{NF}\sum_{t=1}^{NT} v_{ji} \cdot y_{jits} + \sum_{j=1}^{NP}\sum_{k=1}^{NK}\sum_{t=1}^{NT} p_{jk} \cdot w_{jkts} \Big).$$

Constraints

The model's constraints are stated as satisfying dealer requirements at every time period and every scenario:

$$\sum_{i=1}^{NF} z_{jikts} + w_{jkts} = d_{jkts} , \qquad\qquad \forall j,k,t,s .$$

Inventory balance at time period t = 1:

$$x_{jits} + l_{ji} = y_{jits} + \sum_{k=1}^{NK} z_{jikts} , \qquad\qquad \forall j,i,s \quad and \quad t = 1.$$

Inventory balance for t >1:

$$x_{jits} + y_{jit-1s} = y_{jits} + \sum_{k=1}^{NK} z_{jikts} , \qquad\qquad \forall j,i,s \quad and \quad t = 2,...T .$$

Inventory capacity constraint:

$$y_{jits} \le n_{ji} , \qquad\qquad \forall j,i,t,s .$$

Production capacity constraint:

$$x_{jits} \le a_{ji} , \qquad\qquad \forall j,i,t,s .$$

Non-negativity of decision variables:

$$x_{jits} \ge 0 , \; z_{jikts} \ge 0 , \; y_{jits} \ge 0 , \; w_{jkts} \ge 0 , \qquad\qquad \forall j,i,k,t,s .$$

The two-stage stage recourse problem

The two-stage recourse model requires the first stage decision variables (for *t=1*) to be the same for all scenarios. This can be achieved by introducing a set of non-anticipativity constraints. We recall from section 3.2 that *Ξ=[1,2..,8]*, hence the restrictions can be written as:

$$x_{ji1s} = x_{ji1s'} , \qquad\quad \forall j,i \text{ where } s \in [1..7], \text{ and } s' = s+1 ,$$

$$y_{ji1s} = y_{ji1s'} , \qquad\quad \forall j,i \text{ where } s \in [1..7], \text{ and } s' = s+1 ,$$

$$w_{jk1s} = w_{jk1s'} , \qquad\quad \forall j,i \text{ where } s \in [1..7], \text{ and } s' = s+1 ,$$

$$z_{jik1s} = z_{jik1s'} , \qquad\quad \forall j,i,k \text{ where } s \in [1..7], \text{ and } s' = s+1 .$$

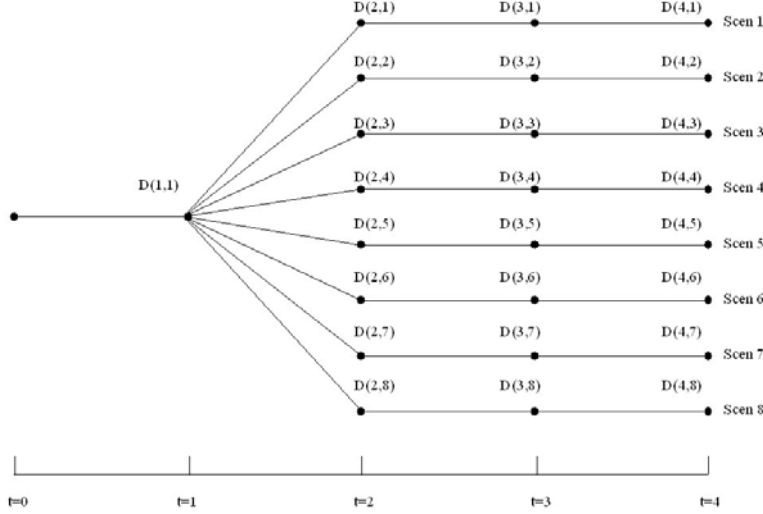These constraints impose on the model the scenario tree structure shown in Figure 4.

Figure 4. Two-stage scenario tree.

Considering the sets of variables:

$$\bar{x}_{ts} = \{x_{jits} \; \forall j, i\}$$
$$\bar{y}_{ts} = \{y_{jits} \; \forall j, i\}$$
$$\bar{w}_{ts} = \{w_{jkts} \; \forall j, k\}$$
$$\bar{z}_{ts} = \{z_{jikts} \; \forall j, i, k\}$$

we can rewrite the above constraints in a more compact notation:

$$D_{11} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{ts} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{ts'} \qquad\qquad t = 1 \text{ where } s \in [1..7] \text{ and } s' = s + 1$$

The non anticipativity restriction forces these sets of decisions to be the same; we can thus substitute them with a named set D_{11}, which represents the decisions associated with the (only) node in the first stage of the scenario tree. In the two stage model, each scenario is independent for *t>1*. The decisions to be taken in the remaining nodes of the tree can be hence defined as:

$$D_{ts} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{ts} \qquad\qquad t > 1 \, , \; s \in \Xi \; .$$

If we use the non-anticipativity constraints to eliminate by substitution the variables that are forced to be the same, we obtain what is often referred to as the *compact* or *implicit* deterministic equivalent representation.

The multistage stage recourse problem

In order to formulate the multi-stage stochastic linear program for the given problem we assume scenarios sharing the same history up to a stage *t* in the scenario tree, must also share the same decisions up to that stage. This means that more non-anticipativity constraints need to be added, so that the tree structure given in Figure 3 can be embodied into the model. The full set of non-anticipativity constraints, expressed in compact notation, is set out below:

$$D_{11} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{1s} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{1s'} \, , \qquad\qquad \text{where } s \in [1..7] \text{ and } s' = s + 1 ,$$

$$D_{21} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{2s} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{2s'}, \qquad \text{where } s \in [1..3] \text{ and } s' = s+1,$$

$$D_{22} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{2s} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{2s'}, \qquad \text{where } s \in [5..7] \text{ and } s' = s+1,$$

$$D_{31} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{3s} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{3s'}, \qquad \text{where } s = 1 \text{ and } s' = 2,$$

$$D_{32} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{3s} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{3s'}, \qquad \text{where } s = 3 \text{ and } s' = 4,$$

$$D_{33} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{3s} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{3s'}, \qquad \text{where } s = 5 \text{ and } s' = 6,$$

$$D_{34} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{3s} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{3s'}, \qquad \text{where } s = 7 \text{ and } s' = 8.$$

In the last stage, a distinct decision is taken for each scenario, therefore we define:

$$D_{4s} = (\bar{x}, \bar{y}, \bar{w}, \bar{z})_{ts} \qquad\qquad t = 4 \ , \ s \in \Xi$$

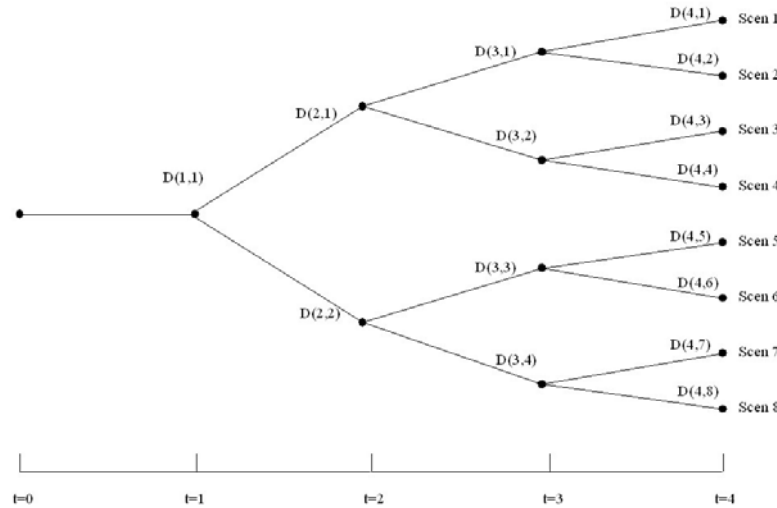The resulting decision tree is shown in Figure 5.

Figure 5. Multistage scenario tree.

4   Adapting algebraic modelling languages for SP

4.1   Alternative modelling approaches

A number of researchers have proposed extensions to modelling systems in order to facilitate the formulation of SP models. For instance, in Gassmann and Ireland, 1995, the authors address the problem of defining scenario based recourse problems using existing AMPL constructs. Scenarios are specified parametrically and the scenario data can either be imported or ideally computed by the AMPL modelling system. The scenario tree structure is represented by first defining a base scenario, and additional scenarios sharing at least the root node with the base scenario are characterised by a parent scenario and the first stage in which the scenario differs from its parent.

In (Fourer 1996), the author proposes some extensions to the AMPL modelling language. These extensions enable the definition of a stochastic programming problem with recourse in terms of a multistage (deterministic) model, a tree of data scenarios for the model, and a stochastic framework

to specify the stages and optionally the scenarios and objective are postulated. New language constructs such as *scenario* and *stochastic* are introduced to enable the definition of a scenario as a collection of data and to declare the partition of an underlying time horizon into stages. Scenarios can be solved individually or as a recourse problem, provided that an appropriate expected value objective is defined. The authors also hint at the possibility of using a new keyword, namely *random*, to assign a probability distribution to given parameters, thus enabling the definition of distribution-based stochastic programming models.

In (Gassmann and Ireland 1996), the authors also propose extension to the AMPL modelling language, mainly for the definition of probability distributions of the random parameters. Again, a language construct *random* is introduced in distribution-based recourse problems to identify the random parameters and the variables which depend on these.

In (Entriken 2001), two additional syntactical items for modelling languages are presented, namely a *random* construct for the definition of random parameters, and a relational operator which indicates precedence between random events. The main idea behind this approach is the fact that stochastic programming models may be seen as control theory problems, where the random events are assumed to be input to the system along with the control variables, so that at a given t, only the past outcomes are known, together with the distribution of the future random parameters. The author uses the syntax of the AMPL language to declare the underlying linear program, and proposes some new constructs for the uncertainty.

An alternative and innovative approach to modelling stochastic linear programming problems is presented in (Buchanan, McKinnon et al. 2001). The authors define a language called sMAGIC, which enables the recursive definition of models which contain other (sub) models. Recursive definition is typical of Dynamic Programming and enables the preservation of the underlying Markov structure, which also characterises many multistage stochastic programming models. The event tree for models with a Markov structure is compactly represented via a special *directed acyclic graph*, which the authors call *Model Link Graph* (MLG).

In (Lopes and Fourer 2004), the authors propose an extension of the AMPL modelling language whereby stochastic models are formulated using a dynamic programming based representation.

4.2   Analysis of modelling issues

Our illustrative example introduced in section 3 highlights the difficulties of using existing algebraic modelling languages to formulate SP. This is mainly due to the lack of constructs for the definition of the randomness of the model coefficients and the scenario tree structure. A stochastic programming model can be considered as a linear programming model extended and refined by the introduction of random parameters (see Figure 6). More precisely, the underlying LP optimisation model is extended by taking into account the probability distribution of the model's random parameters. Such distributions are provided by the models of randomness used in *scenario generators*, which are specific to the particular optimisation problems under investigation.
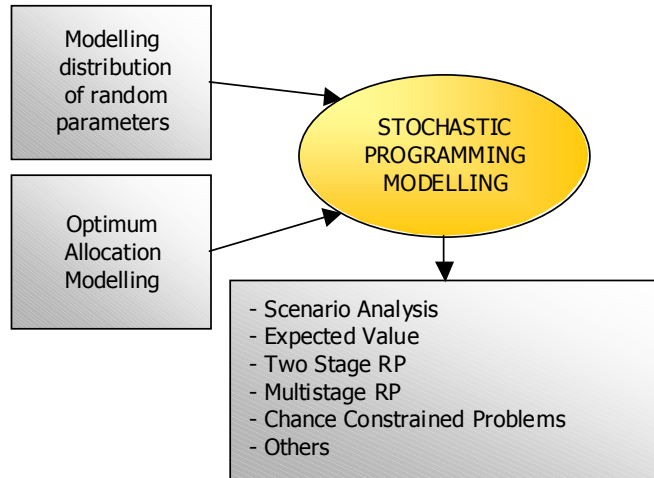
In general, different categories of stochastic programming problems require different language features to express the random nature of the problem. We call *stochastic framework* the information represented by these constructs.

Scenario based recourse problems

The first requirement for the formulation of a stochastic programming problem using algebraic modelling languages is the declaration of the random parameters. In scenario based recourse problems, the realisations of such parameters are explicitly given in the form of a scenario tree. Each scenario is also associated with a corresponding weight (or probability). In turn, the scenario tree structure is declared in terms of stages. The stages identify the sequence of decisions in the dynamics of the underlying core model. If the temporal dimension is introduced into the model using a specific time set, the stages can be declared as subsets of this set. To summarise, the stochastic framework for scenario based recourse problems requires constructs for the definition of stages, scenarios and random parameters (see Table 1).

Entities	Language Requirements
stages information	assignment of variables and constraints to stages.
scenarios information	scenarios set
	tree structure
	scenario probabilities
random parameters	declaration of the random parameters in terms of the scenario set.

Table 1. Requirements for scenario based recourse problems

Modelling distribution based recourse problems

A number of researchers have proposed extensions to algebraic languages for the formulation of such class of problems (Fourer 1996; Gassmann and Ireland 1996; Fourer 2001; Gay 2001).

In this work, we focus on the class of scenario-based recourse problems and only outline the requirements for distribution-based models. Distribution based recourse problems rely on the declaration of probability distributions for the random parameters. This can be discrete or continuous. If all random parameters are characterised by discrete distributions, the scenario tree is

implied by the joint realisations of the random parameters. If one or more distributions are continuous, then there are infinite many possible outcomes for the random parameters and the tree structure must be sampled from the joint distributions. The stochastic framework for distribution based problems requires constructs shown in Table 2.

Entities	Language Requirements
stages information	assignment of variables and constraints to stages.
random parameters	declaration of the probability distributions associated with the random parameters.

Table 2. Requirements for distribution based recourse problems

4.3 Overview of our approach

We present a generic approach of expanding AMLs based on the concepts of *underlying deterministic model* and *stochastic framework*. The underlying deterministic model is formulated using the standard constructs provided by algebraic modelling languages. Using a set of new constructs, the modeller then declares the stochastic framework, which links the underlying deterministic model with the model of randomness. More specifically, the underlying deterministic model represents a family of independent (wait and see) models, while the stochastic framework imposes the non anticipativity implied by the structure of the scenario tree. The resulting model is the formulation of the stochastic programming recourse problem. Chance constraints can also be introduced using specific constructs.

The underlying deterministic model

In a typical stochastic programming problem, it is always possible to identify an underlying deterministic model (also called the *core* model). This can be the expected value problem or a problem corresponding with any sample path of the scenario tree. The underlying deterministic model captures the logical structure of the problem as well as the dynamical relations within decision variables, their bounds and the objective function.

In our approach, we construct the core model in such a way that this is parametric in the dimension of scenarios. All variables and constraints are indexed over the scenarios (which are the element of a special set declared in the stochastic framework), and the objective function is the expected value over all scenarios of the individual objectives. As previously mentioned, this formulation is equivalent to a family of wait and see models.

Declaration of the stochastic framework

As we have shown in section 4.2, the stochastic framework depends on the type of stochastic programming model which is being developed. For instance, scenario based recourse problems require the explicit declaration of the scenario tree structure, while in a distribution based recourse problem, the AML should provide a set of constructs for the definition of probability distributions.
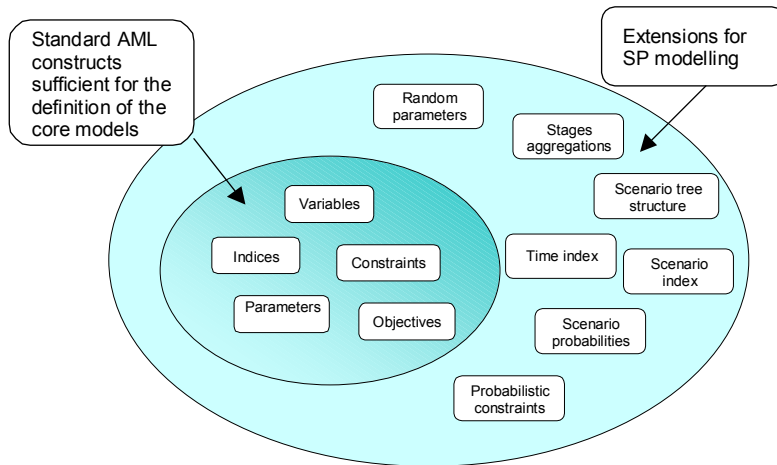
Figure 7. Extended language constructs

Figure 7 shows how the basic constructs of a modelling language for linear programming are extended to capture the stochastic framework. The definition of the new constructs is adapted to be consistent with the grammar of the underlying modelling language. We have successfully applied this approach to the AMPL language, although the same ideas can be adapted to virtually any other AML, see for instance (Valente, Mitra et al. 2005). The syntax of the extended language constructs for AMPL (which we call SAMPL) is defined in the next section.

4.4   SAMPL language extensions

Taking into account the requirements expressed in section 4.2, we have designed a set of new constructs for the AMPL language, which support the declaration of the stochastic framework information.

Stages

The assignment of variables and constraints to the different stages of the dynamic recourse model is essential in the definition of stochastic programming recourse models.

A generic approach considers the existence of a *time* set, which is used as an index for all variables of the model. Then a partition of such time set can be given, so that several time periods (and the variables which relate to these time periods) can be grouped into a single stage. We have implemented this approach in extending the MPL modelling language (Valente, Mitra et al. 2001). However, the fact that modellers are forced to introduce the time dimension for all variables could result in a very unnatural modelling. This is particularly true in two stage models, where the first stage and second stage decisions are usually defined using different vectors, and hence already "distinguished" from one another.

An alternative and more flexible approach is viable using modelling languages (such as AMPL), which enable the definition of *suffixes*. A suffix can be considered as a generic property of a variable or constraint, and can be used for our purpose to declare the stage which a variable belongs to. The stage of the constraints is determined by the stage of the variables which appear in it. The highest stage of any of such variables is the stage of the constraint.

The syntax for the assignment of a stage number to a variable is very similar to that of AMPL for other suffixes, but makes use of a predefined suffix called *stage*:

```
suffix stage IN;    #AMPL keyword to declare a suffix called stage
let indexing_opt  name.stage := expr;
```

Alternatively, AMPL enables the suffix to be given in the variable declarations:

var name alias_{opt} indexing_{opt} attributes_{opt}, **suffix stage** *expr;*

In the example distribution model, variables are indexed over a time set. The staging for the two-stage problem can be expressed as:

```
suffix stage IN;
var x{Prod,Fact,t in Time,Scen}   >=0, suffix stage if t=1 then 1 else 2;
var y{Prod,Fact,t in Time,Scen}   >=0, suffix stage if t=1 then 1 else 2;
var w{Prod,Deal,Time, Scen }      >=0, suffix stage if t=1 then 1 else 2;
var z{Prod,Fact,Deal,Time, Scen}  >=0, suffix stage if t=1 then 1 else 2;
```

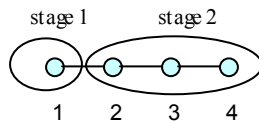Figure shows the aggregation of time periods into stages for the two-stage distribution model.

Figure 8. Two-stage aggregation.

In the multistage model, the stage of a variable is simply given by the time period which the variable belongs to:

```
suffix stage IN;
var x{Prod,Fact, Time,Scen}      >=0;
var y{Prod,Fact, Time,Scen}      >=0;
var w{Prod,Deal,Time, Scen }     >=0;
var z{Prod,Fact,Deal,Time, Scen} >=0;

let {p in Prod, f in Fact,t in Time,s in Scen}           x.stage=t;
let {p in Prod, f in Fact,t in Time,s in Scen}           y.stage=t;
let {p in Prod, d in Deal,t in Time,s in Scen}           w.stage=t;
let {p in Prod, f in Fact,d in Deal,t in Time,s in Scen} z.stage=t;
```

Figure shows the aggregation of time periods into stages for the multistage distribution model.
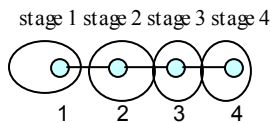
Figure 9. Multistage aggregation.

Scenarios information

Information about the scenario tree which characterises scenario-based recourse problems is provided by the modeller via a set of new keyword, which describe below:

Scenario set

In scenario-based recourse problems, the uncertainty represented by the random parameters introduces a new dimension, identified by the scenario set. This set needs to be explicitly identified, because the random parameters are indexed over it. The syntax used for the declaration of the scenario set follows the syntax of AMPL for sets, but uses *scenarioset* instead of *set* in the declaration:

scenarioset *name alias_{opt} indexing_{opt} attribs_{opt} ;*

Considering the example model given in section 3, the scenario set is declared as:

```
param NS:=8;
…
#stochastic framework
…
scenarioset Scen:=1..NS;
…
```

Scenarios probabilities

The probability distribution of the scenarios is declared using a special parameter vector, whose elements are the weights of the individual scenarios. The parameter is therefore indexed over the scenario set. The syntax used is similar to the AMPL syntax for the declaration of the other parameters, but uses the *probability* keyword instead of (or before) the *param* keyword. The syntax is the following:

probability param$_{opt}$ name alias$_{opt}$ indexing$_{opt}$ attributes$_{opt}$;

Example:

```
…
scenarioset Scen:= 1..NS;

probability param Pr{Scen}:=1/card(Scen);
…
```

Scenario Tree

The *tree* keyword is used to define the scenario tree structure. In general, this structure can be considered as a set of pairs (scenario,stage), which identify the branches (and therefore the nodes) of the tree. Our language constructs enable the modeller to specify "well structured" trees in a concise way. The syntax is as follows:

tree name:=$_{opt}$ tree_declaration ;

where *tree_declaration* is one of:

bundle_list |
tlist |
nway{n}|
multibranch{n$_1$, n$_2$,...,n$_{ST}$}|
binary |
twostage {ns}$_{opt}$;

- **bundles**

A bundle list is defined as:

bundle_list : bundles{Bundle-1, Bundle-2,.. Bundle-n} ;

where:

Bundle-x: (stage$_x$,scen$_x$);

A bundle is associated to each node of the tree and consists of the stage in which the node lies, and a scenario number defined as the minimum of the ordinal values associated with the scenarios which pass through the same node (King 1994). Considering the tree structure of Figure 10, its bundles representation is formulated as:

```
…
tree theTree:=
bundles
{
(1,1),
(2,1),(2,4),(2,6),
```

```
(3,1),(3,4),(3,6),(3,7),
(4,1),(4,2),(4,3),(4,4),(4,5),(4,6),(4,7)(4,8),(4,9)
};
```
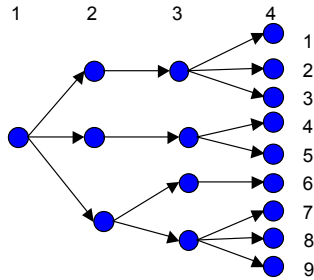

Figure 10. Example of asymmetric tree.

- **`tlist`**

An very compact representation of asymmetric trees can be obtained if the following assumption holds:

Scenarios are indexed in an order that ensures complete separation of all nested sub-trees (i.e. the scenario paths never cross one another)

In this case, a scenario tree can be represented as a list of S elements (*tlist*) whose values are the stage numbers where a scenario branches from its father scenario. A *tlist* can thus be constructed by considering the column number associated to the leftmost element of each row of the tree matrix. The syntax for the tlist tree is the following:

tlist: **`tlist`**$\{n1, n2,.. n_S\};$

As an example, the tlist representing the scenario tree of Figure is set out below:

```
…
tree theTree:= tlist {1,4,4,2,4,2,3,4,4};
…
```

In many applications, however, the scenario tree is characterised by a "standard" structure and SAMPL provides simplified syntax to specify it. For a discussion of the alternative ways of specifying the scenario tree, the reader should refer to (SPInE/SAMPL user guide, 2005).

Random parameters

The random parameters of the model need to be specified so that these can complement the deterministic model definition. For the scenario based model, every parameter of the model which is random, needs to be explicitly identified as such. Also, the scenario dimension must be added to the indexing list of the parameter.

random param *name indexing attributes$_{opt}$;*

For the example of section 3.1, the deterministic demand parameter for the EVLP is defined as:

```
…
param d{Prod,Deal,Time,Scen }     >=0;    #demand
…
```

Using the *random* keyword, we explicitly redefine the parameter as a random variable:

```
scenarioset Sc:=1..NS;
…
random param d{Prod,Deal,Time,Scen} >=0;      #demand
```

The particular nature of the multistage stochastic programs with recourse introduces the choice between diverse forms of presenting the data values for the random parameters. These forms can be either *compact* or *expanded*. The compact form assumes that the scenario data can present itself in the form of a 2-dimensional matrix (tree matrix) where only some of the entries exist. The matrix forms a grid where the columns represent time periods and the rows represent scenarios. The conditions under which such a matrix, partially entered, can represent a tree may be stated as follows:

1. Entry *(1,1)* must exist

2. Entries *(T,s)* exist for all scenarios *s=1..S*, where *T* is the number of time periods and S the number of scenarios.

3. If entry *(t,s)* exists for some *t<T*, then also does entry *(t+1,s)*.

Each entry *(t,s)* of the matrix represents the realisation of the random parameter at time period t under scenario s. An entry can consist of a single value or a vector of values if the random parameter is indexed over sets other than the time set and the scenario set.

For instance, let us consider a 3 time period horizon and random parameter *p*, which takes the (known) value *10* in the first time period. Let us consider that *2* realisation of the random parameter can be observed at each time period *t=2..4*, as in figure Figure 3:
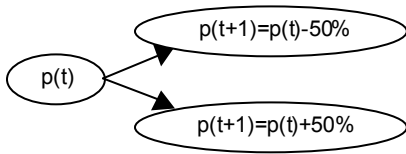
Figure 3. Simple event tree rule.

This rule defines a simple scenario generator, whereby a binary tree can be constructed, as in figure Figure .
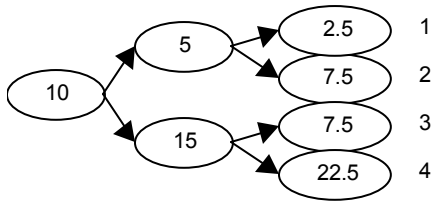
Figure 12. Binary scenario tree.

The (sparse) matrix representing the data for this scenario tree is shown in Table 3:

t=1	t=2	t=3	
10	5	2.5	s=1
		7.5	s=2
	15	7.5	s=3
		22.5	s=4

Table 3.Compact scenario data.

Such matrix can be represented in SAMPL in a row-wise fashion as:

```
#model file
```

```
scenarioset scen = 1..4;

random param dem{t,scen};
…
#data file

random param dem:=
1 1                         10
2 1                          5
2 3                         15
3 1                        2.5
3 2                        7.5
3 3                        7.5
3 4                       22.5;
```

4.5   Implementation of the illustrative examples in SAMPL

In this section, we combine the AMPL standard constructs with the proposed SAMPL extension in order to formulate the illustrative example of section 3 in its two-stage and multistage versions. More specifically, we first formulate the underlying deterministic model (Wait and See) in AMPL, and then add the stochastic information using the SAMPL constructs.

The underlying deterministic model

The underlying deterministic model associated with the example problem can be formulated in AMPL as follows:

```
#distribution model clo2s.mod

param NT; param NK; param NP; param NF; param NS;

set Fact:= 1..NF;
set Prod:= 1..NP;
set Deal:= 1..NK;
set Time:= 1..NT;
set Scen:= 1..NS;

param q{ Prod, Fact };              #unit prod cost
param c{ Fact,Deal};                #unit transportation cost
param v{ Prod, Fact };              #prod capacity
param a{ Prod, Fact };              #inventory cost
param l{Prod,Fact};                 #Initial inventory
param n{Prod,Fact};                 #Inventory capacity
param p{Prod,Deal};                 #Shortage penalty

param d{Prod,Deal,Time,Scen};       #demand
param Pr{Scen}:=1/card(Scen);       #Probability of the scenarios

var x{Prod,Fact,Time,Scen}        >=0;    #production
var y{Prod,Fact,Time, Scen }      >=0;    #inventory
var z{Prod,Fact,Deal,Time, Scen}  >=0;    #shipment
var w{Prod,Deal,Time, Scen }      >=0;    #shortage

minimize cost: sum{s in Scen} Pr[s]*
            (sum{j in Prod, i in Fact, t in Time}           q[j,i]*x[j,i,t,s] +
            sum{j in Prod, i in Fact, k in Deal, t in Time}  c[i,k]*z[j,i,k,t,s] +
            sum{j in Prod, i in Fact, t in Time}            v[j,i]*y[j,i,t,s] +
            sum{j in Prod, k in Deal, t in Time}            p[j,k]*w[j,k,t,s]);

subject to

satisfy_demand{j in Prod, k in Deal, t in Time, s in Scen}:
            sum{i in Fact} z[j,i,k,t,s] + w[j,k,t,s]=d[j,k,t,s];
```

```
inv_balance_init{j in Prod, i in Fact, s in Scen }:
                x[j,i,1,s]+l[j,i]=y[j,i,1,s]+sum{k in Deal} z[j,i,k,1,s];

inv_balance{j in Prod, i in Fact, t in 2..NT, s in Scen }:
                x[j,i,t,s]+y[j,i,t-1,s]=y[j,i,t,s]+sum{k in Deal} z[j,i,k,t,s];

inv_capacity{j in Prod, i in Fact, t in Time, s in Scen }:
                y[j,i,t,s]<=n[j,i];

prod_capacity{j in Prod, i in Fact, t in Time, s in Scen }:
                x[j,i,t,s]<=a[j,i];
```

The two stage recourse model

To illustrate the use of the new constructs for stochastic programming described in the previous section, we modify the above deterministic model as follows:

```
scenarioset Scen:=     1..NS;          #declares Scen as the "special" scenario set;
tree        theTree:=  twostage{NS};   #declares a two stage tree with NS scenarios

random param d{Prod,Deal,Time,Scen};     #declares demand as random parameter
probability param Pr{Scen}:=1/card(Scen);#declares the probability of the scenarios

suffix stage IN;                         #assigns variables to stages

var x{Prod,Fact,Time,Scen}        >=0, suffix stage if t=1 then 1 else 2;
var y{Prod,Fact,Time,Scen}        >=0, suffix stage if t=1 then 1 else 2;
var z{Prod,Fact,Deal,Time,Scen}   >=0, suffix stage if t=1 then 1 else 2;
var w{Prod,Deal,Time,Scen}        >=0, suffix stage if t=1 then 1 else 2;
```

The set Scen is defined using the *scenarioset* keyword, which identifies it as a special set. The *tree* keyword is used to define the structure of the scenario tree as a two-stage (fan) tree.

The parameter d is declared using the *random param* keyword, while the probability vector Pr is redefined as *probability param*. Finally, the variables are partitioned into different stages by adding of the suffix *stage* to their definition.

From two stage to multi stage

Transforming the two-stage model to a multistage formulation for the same problem is very easily achieved using the SAMPL constructs. It is necessary to disaggregate the stages, and assign variables of different time periods to distinct stages. The tree structure of the model is also revised to that of the multistage tree of Figure . The resulting stochastic framework is set out as follows:

```
scenarioset Scen:=     1..NS;        #declares Scen as the "special" scenario set;
tree        theTree:=  nway{5};      #declares tree with 5 branches at each stage

random param         d{Prod,Deal,Time,Scen};  #declares demand as a random parameter
probability param    Pr{Scen}:=1/card(Scen);  #declares the probability of the scenarios

suffix stage IN;                              #assigns variables to stages
var x{Prod,Fact,Time,Scen}          >=0;
var y{Prod,Fact,Time,Scen}          >=0;
var z{Prod,Fact,Deal,Time,Scen}     >=0;
var w{Prod,Deal,Time,Scen}          >=0;

let {p in Prod, f in Fact,t in Time,s in Scen}          x.stage=t;
let {p in Prod, f in Fact,t in Time,s in Scen}          y.stage=t;
let {p in Prod, d in Deal,t in Time,s in Scen}          w.stage=t;
let {p in Prod, f in Fact,d in Deal,t in Time,s in Scen} z.stage=t;
```

5 Implementation of the language extensions

Algebraic modelling languages are supported by *modelling systems*, which provide connectivity to data sources and solvers. In order to enhance this with features which support stochastic programming constructs, some key aspects need to be taken into account:

- The inadequacies of existing matrix representation formats for SP problems, used to connect solvers to the modelling systems.

- The problem of linking modelling systems with scenario generators.

In this section, we address the first two issues and briefly introduce our SPInE system, which implements the language extensions presented in this paper and is readily connected with SP solvers and scenario generators.

5.1 Interfacing modelling systems and solvers

The main consideration for the interfacing of modelling system and specialised SP solvers is the representation and communication of the SP model instances.

Representation of SP model instances.

The SMPS format as introduced in (Birge, Dempster et al. 1988) enables the compact representation of large scale stochastic programming problems, including scenario based recourse problems. However, in (Gassmann and Schweitzer 2001) the authors illustrate a number of shortcomings associated with this standard. The authors propose an extended SMPS format with added representational power which overcomes most of the problems associated with the original standard. In this research, however, other types of problems have been identified, which cannot be directly represented using the SMPS standard, nor its extended version.

Recourse problems without first stage constraints.

Consider the following simple problem, which is a variant of the newsboy problem:

The newsboy needs to decide, every day, what is the optimal number of newspaper x to buy in order to maximize the profits. The demand D is not known with certainty. A set of scenarios S for the demand is given. The outcomes of the demand for each scenario s in S are defined as D_s and the associated probability is P_s. Each copy of the newspaper is characterized by a unit selling price R and a unit purchase cost C. Unsold copies y_s^+ represent a loss for the newsboy. The eventual shortfall of copies is indicated with y_s^- and is characterised by a penalty term in the objective function. Shortfall and excess are only revealed once the demand is known, hence x is a *first stage variable*, while y_s^+ and y_s^- are *second stage variables*. The algebraic formulation of the two-stage model can be stated as follows:

$$\max \; profit = (R - C)x - (R - C)\sum_{s \in S} P_s y_s^- - R\sum_{s \in S} P_s y_s^+ \, ,$$

subject to

$$x + y_s^- - y_s^+ = D_s \quad \forall s \in S.$$

(18)

where the objective is to maximise the profit, expressed as the profit of selling all the newspaper purchased, minus the expected value of the (false) profit generated by the unsold copies and the expected value of the shortfall penalty. The ordered matrix of this problem, for $|S|=3$ is the following:

X	y_1^-	y_1^+	y_2^-	y_2^+	y_3^-	y_3^+	
R-C	P_1(R-C)	P_1R	P_2(R-C)	P_2R	P_3(R-C)	P_3R	
1	1	-1					D_1
1			1	-1			D_2
1					1	-1	D_3

If the first stage problem is unbounded, solution algorithm on Benders' decomposition may fail. In general, these algorithms try to solve a master problem associated with the first stage in order to obtain a feasible first stage solution, which is then refined by solving sequences of subproblems (associated with second stage) which are parametric in the first stage decision.

Global constraints

The term "global constraint" is used in the context of this research to identify a class of constraints which may be required in some scenario based stochastic programming problems. A global constraint is a constraint which involves decision variables associated with different nodes of the scenario tree. Consider for instance the following two-stage recourse problem:

$$\min cx + \sum_{s \in S} p_s f_s y_s \,,$$

subject to

$$Ax = b,$$ (19)

$$u_s x + t_s y_s = h_s \quad \forall s \in S,$$

$$\sum_{s \in S} p_s r_s y_s \leq k.$$

The associated matrix for $|S|=3$ is the following:

x	y_1	y_2	y_3	
c	$p_1 f_1$	$p_2 f_2$	$p_3 f_3$	
a				b
u_1	t_1			b_1
u_2		t_2		b_2
u_3			t_3	b_3
	$p_1 r_1$	$p_2 r_2$	$p_3 r_3$	k

The last row links variables belonging to three different scenarios. One of the underlying assumptions of SMPS is that the blocks of the matrix which relate to different scenarios need to be completely separable, therefore these types of problems cannot be represented using the SMPS format. Global constraints are generally used to introduce restrictions on the statistical distributional properties of the decision taken in the first stage and evaluated for all the discrete scenarios. Typical examples of such functions are risk measures such as the *Expected Downside Risk* or the *Conditional Value at Risk* (CVaR).

A modelling system which supports stochastic programming modelling should therefore take into account the issues related to such extensions to the representation of the model instances.

New directions: XML

The limitations of the SMPS format in terms of representational power have become apparent over the years, and some extensions have been proposed (Gassmann and Schweitzer 2001). Recently, researchers have expressed significant requirements for an enhanced standard, specifically for the purpose of communicating mathematical programs. A new approach, which takes into consideration contemporary technical assumptions, is based on *eXtended Mark-up Language* (XML). The wealth of functionality and software available for exploiting XML is an important factor in the acceptance of an XML based mathematical programming standard. There are currently at least three XML vocabularies proposed for this role: FML (Fourer, Lopes et al. 2003), OptML (Kristjansson 2001) and SNOML (Lopez and Fourer 2001). Both support linear and non-linear optimisation models, but in addition, SNOML is also able to capture stochastic programming models with recourse, chance-constrained models, and also constraint logic programming models.

At this stage, OptML and SNOML are not fully designed nor accepted by the research community; however, they seem to be the most promising approaches towards a new standard for the representation and communication of mathematical programming problems, including stochastic programming models.

5.2   Integrated environments: SPInE

Our implementation of the SAMPL language processor is the core of the modelling subsystem of our Stochastic Programming Integrated Environment (SPInE) (Valente, Mitra et al. 2005).

SPInE is divided into four main subsystems, namely Scenario Generation, Modelling, Solver, Results Analysis and the overarching Control module.
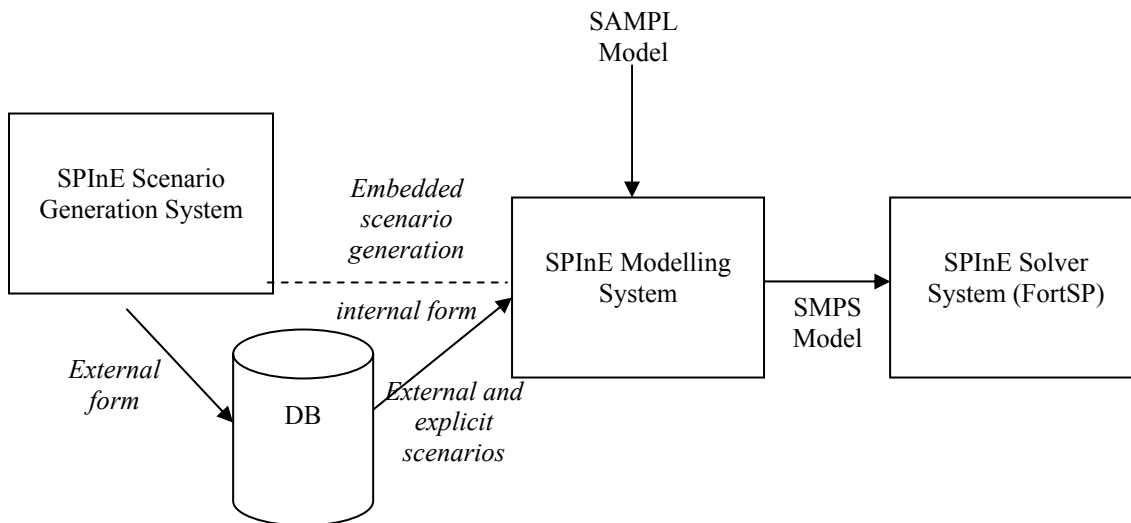

Figure 13. SPInE Architecture

Scenario Generation

SPInE is designed to interface with scenario generators which supply the scenario data in ODBC databases or text files. An important aspect of the scenario generation interface is to establish the

consistency between the SP model tree and the data path tree underlying the scenario generation. A possible architecture for implementing a direct communication between modelling system and scenario generator has been studied by Di Domenica, et al (2006) and is currently under development. The design of the system includes automatic check for the consistency of the generated data.

Modelling subsystem

The modelling subsystem is designed to support the SAMPL language extensions introduced in this paper. A software module called *Stochastic Program Generator* (SPG) combines two separate parsers and a matrix generator. SPG processes together the algebraic models and the scenario data set to create an instance of the model in either SMPS format or in the Stochastic Intermediate Representation (SIR). The SPG module makes use of an underlying *modelling engine*, specifically AMPL-based COM object, developed by our research group, for models prepared in SAMPL. The modelling system interacts directly with the scenario generators for the stochastic data can connect to database systems or data files which store the deterministic data relating to the core model of the SP problem.

Solver subsystem

Given a Stochastic Programming problem with recourse, formulated in SAMPL, the FortSP stochastic solver embedded in SPInE (Poojari, Ellison et al. 2002) provides the solution to three related classes of models:

- Here and Now

- Scenario Analysis (Wait and See)

- Expected Value

For each of these, there is more than one possible *solution algorithm*. All underlying LPs may be solved using the Sparse Simplex algorithm (SSX) or the Interior Point Method (IPM). The Here and Now problem may be solved using Benders Decomposition, Lagrangean Relaxation or via the Deterministic Equivalent problem. The solver is also organised to compute the stochastic measures EVPI and VSS.

Results analysis

A critical phase in the development of stochastic models is the analysis of the solutions. The integration with database systems enables the exploitation of the Data Manipulation Languages (DML) which usually accompanies the DBMS for the development of customised viewers and advanced data analysis tools. The SP Reporter (SPR) module of SPInE allows the user to export solution vectors using standard ODBC or using text files. The volume of the solution results produced by the stochastic solver can be very large. In fact, each decision variable has an associated optimal activity and reduced cost, for each stage and for each scenario. The investigator might be interested only in a subset of the solutions (e.g. the first stage strategic decisions). SPR provides *filtering* functionality which is used to transfer only the relevant decision data to the DBMS.

Control Module and Graphical User Interface

Each module in SPInE can be run as an independent application through script files. A control module including a Graphical User Interface (GUI) has been developed and can be used to investigate SP problems. The main subsystems of SPInE, namely the SP instances generator SPG, the stochastic solver FortSP and the solution Reporter SPR have also been wrapped in a dynamic link library, which enables the rapid development of embedded applications.

6 Conclusions and directions for future research

Modelling and solving stochastic programming problems is a challenging task. Stochastic programming models can be transformed into deterministic equivalent formulations by discretising the support of underlying continuous probability distributions. However, it is often impossible to take advantage of the existing modelling tools and LP/MIP solvers to formulate and process these models. In this paper, we have identified a number of modelling requirements for stochastic programming. Taking these into account, we have designed an approach to extending algebraic modelling languages for the formulation of SP models in a natural and concise way and we have shown how this approach can be applied to the AMPL modelling language with a practical example.

The difficulties associated with the modelling of the random parameters for SP problems are discussed, and some insight is gained into the relations between scenario trees and SP model structures; thereby the requirements for the seamless integration of scenario generators and modelling systems are identified. Current algebraic modelling languages are required to capture the scenario structures of SP models. The problem of representing instances of stochastic programming models in formats which can be exploited by specialised solution algorithms are also studied.

A number of issues which remain to be addressed have been identified in respect of the process of modelling, solution and analysis of stochastic programming problems.

The language constructs proposed in this research address a limited but important class of scenario-based stochastic programming problems. The definition of the random parameters using probability distributions needs to take into account the complexity introduced, for instance, in multistage problems. In these models, there might be dependency of the random variables across stages, which implies the use of conditional probabilities.

The integration of scenario generators requires further investigation. This aspect is strictly connected with the possibility of defining models of randomness directly in the modelling language. A library of standard scenario generators which readily connects to the SPInE system is under development. This will enable practitioners to truly deploy the system for a complete investigation of stochastic programming problems, and will also expand the capabilities of a modelling system which combines optimum decision modelling and descriptive simulation modelling, see DiDomineca et al (2007).

7   References

Birge, J. R., M. A. H. Dempster, H. I. Gassmann, E. Gunn, A. J. King and S. W. Wallace (1988). "A standard input format for multiperiod stochastic linear programs." <u>Mathematical Programming Society Committee on Algorithms Newsletter</u> **17**, *1-19.*

Birge, J. R. and F. Louveaux (1997). "Introduction to stochastic programming." <u>Springer Verlag,</u> New York.

Bisschop, J. J. and M. Roelofs (1999). "AIMMS: The Language Reference." <u>Paragon Decision Technology BV</u>, Haarlem, The Netherlands.

Bosetti, V., E. Messina and P. Valente (2002). "Optimisation technologies and environmental applications." <u>submitted to the IMA Journal of Management Mathematics</u>.

Brooke, A., D. A. Kendrick, A. Meeraus and R. Raman (1998). "GAMS : a user's guide,Release 2.50."

Buchanan, C. S., K. I. M. McKinnon and G. K. Skondras (2001). "The Recursive Definition of Stochastic Linear Programming Problems within an Algebraic Modeling Language." <u>Annals of Operations Research</u> **104**(1), *15-32.*

CARISMA, Lecture Notes(2005) on Optimization Under Uncertainty, Stochastic Programming,

CARISMA , Brunel University, Uxbridge.

Charnes, A. and W. W. Cooper (1959). "Chance-constrained programming." <u>Management Science</u> **5**, *73-79.*

Di Domenica, N.. Mitra, G., Valente, P., (2006) Stochastic programming and scenario generation: an information systems perspective, to appear in Decision Support Systems.

Di. Domenica, N., Mitra, G., Lucas, C.,(2007) Stochastic Programming and Simulation : a modelling perspective, to appear in IMA Journal of Management Mathematics..

Entriken, R. (2001). "Language Constructs for Modeling Stochastic Linear Programs." <u>Annals of Operations Research</u> **104**(1), *49-66.*

Eppen, G. D., R. K. Martin and L. Schrage (1989). "A Scenario Approach to Capacity Planning." <u>Operations Research</u> **37**(4), *517-527.*

Escudero, L. F., E. Galindo, G. Garcia, E. Gomez and V. Sabau (1999). "Schumann, a modeling framework for supply chain management under uncertainty." <u>European Journal of Operational Research</u> **119**(1), *14-34.*

Fourer, R. (1996). "Proposed new AMPL features - stochastic programming extensions." <u>www.ampl.com/cm/cs/what/ampl/NEW/FUTURE/stoch.html.</u>

Fourer, R. (2001). "Model-level directives for generating alternative formulations of stochastic programs." <u>IX International Conference on Stochastic Programming,</u>, Berlin, Germany.

Fourer, R., D. M. Gay and B. W. Kernighan (1993). "AMPL : a modeling language for mathematical programming." <u>Scientific press, San Francisco, CA</u>

Fourer, R., L. Lopes and K. Martin (2003). "A W3C XML schema for linear programming." <u>Northwestern University and The University of Chicago</u>, (work in progress).

Fourer, R. and Lopes, L.B. (2004). "StAMPL: A Filtration-Oriented Modeling Tool for Stochastic Programming." Technical report 04-008, <u>Department of Industrial Engineering and Management Sciences, Northwestern University</u>, Evanston, IL.

Gassmann, H. I. and A. M. Ireland (1995). "Scenario formulation in an algebraic modelling language." Annals of Operations Research **59** *45-76*.

Gassmann, H. I. and A. M. Ireland (1996). "On the formulation of stochastic linear programs using algebraic modelling languages." Annals of Operations Research **64**, *83-112*.

Gassmann, H. I. and E. Schweitzer (2001). "A comprehensive input format for stochastic linear programs." Annals of Operations Research **104**(1), *89-125*.

Gay, D. M. (2001). "Random parameters - a step toward conveniently expressing some stochastic programming problems." IX International Conference on Stochastic Programming, Berlin, Germany.

Higle, J. L. and S. Sen (1996). Stochastic Decomposition, Kluwer Academic Publishers, Dodrecht / Boston / London

Jobst, N. (2002). On credit risk modelling, simulation, optimisation. PhD thesis, Department of Mathematical Sciences. Uxbridge, UK, Brunel University.

Kall, P. and J. Mayer (2005). "Building and solving stochastic linear programming models with SLP-IOR." in S. W. Wallace and W. T. Ziemba (Eds): Applications of Stochastic Programming, 79-94

King, A. J. (1994). "SP/OSL Version 1.0 Stochastic Programming Interface Library: User' s Guide, Research Paper RC 19757 (87525) 9/26/94 Mathematics." T.J Watson Research Center, Yorktown Heights, NY.

Kristjansson (2001). "How New Technologies such as XML and SOAP allow OR to provide Web-based Services." INFORMS Roundtable Savannah, Georgia.

Lopes, L. and R. Fourer (2003). "A modeling language for stochastic programming." ISMP 2003, Copenhagen, Denmark.

Lopez, L. and R. Fourer (2001). "An XML-Based Format for Sharing Mathematical Programs." INFORMS international conference on Miami, November 2001.

Lucas, C., E. Messina and G. Mitra (1997). "Risk and Return Analysis of a Multi-period Strategic Planning Problem." Lecture notes in economics and mathematical systems (445), *81*.

Maximal Software (2002). "MPL Modelling System, Release 4.2." USA.

Messina, E. and G. Mitra (1997). "Modelling and analysis of multistage stochastic programming problems: A software environment." European Journal of Operational Research **101**(2), *343-359*.

Pereira, M. and L. Pinto (1985). "Stochastic Optimization of a Multireservoir Hydroelectric System: A Decomposition Approach." Water Resources Research **21**(6), *779-792*.

Poojari, C. A., E. F. D. Ellison and G. Mitra (2002). "A multi-stage stochastic programming solver." Working paper, CARISMA Centre for the Analysis of Risk and Optimisation Modelling Applications, Brunel University, London, UK.

Shapiro, J. (2001). "Modeling the Supply Chain." Duxbury Press, California.

SPInE manual, (2004)

http://www.optirisk-systems.com/documents/SpineAmpl/PDF/SpineAmplManual.pdf

Valente, P. (2002). Software tools for the investigation of stochastic programming problems. PhD Thesis, Department of Mathematical Sciences. Uxbridge, UK, Brunel University.

Valente, P., G. Mitra and C. A. Poojari (2005). "A Stochastic Programming Integrated Environment (SPInE)." in S. W. Wallace and W. T. Ziemba (Eds): Applications of Stochastic Programming, 115-136.

Valente, P., G. Mitra, C. A. Poojari and T. Kyriakis (2001). "Software Tools for Stochastic Programming: A Stochastic Programming Integrated Environment (SPInE)." TR 10/01, <u>Department of Mathematical Sciences, Brunel University</u>, London, UK.

Ziemba, W. T. and J. M. Mulvey, (Eds). (1998). "Worldwide asset and liability modeling." Cambridge University Press, Cambridge.