Annals of Operations Research 64(1996)83–112

On the formulation of stochastic linear programs using algebraic modelling languages

H.I. Gassmann and A.M. Ireland
School of Business Administration, Dalhousie University, Halifax, Nova Scotia, Canada B3H 1Z5
E-mail: gassmann@earth.sba.dal.ca; aireland@ac.dal.ca

This paper considers extensions to algebraic modelling languages to support formulation, instantiation and solver integration for stochastic linear programs (SLPs). We present a taxonomy of SLP problem types and analyze formulation requirements including distribution handling by class of problem. We demonstrate suggested formulations for most problem classes, show solver input in the S-MPS standard, and propose consistency checks for constraints involving stochastic data items. Some unresolved difficulties are identified.

1 Introduction

Algebraic modelling languages (AMLs) for mathematical programming are languages in which mathematical programming models can be stated using language constructs which closely resemble mathematical notation [21,32]. These languages assist modellers in the formulation, solution and management of large mathematical programs by providing powerful, declarative algebraic syntax for model specification, often in systems with integrated LP solvers and a range of model management facilities. Their capabilities have reduced the programming skill required for modellers to implement large LPs, encouraging wider application of linear programming in practical situations.

Use of stochastic linear programming (SLP) models is becoming more widespread as computing capacity and solver technology rapidly improve and as SLP advantages are demonstrated (see for example, Cariño et al. [8], Mulvey and Vladimirou [36], Dempster and Ireland [13], Sen et al. [44]). However, SLP modellers now face difficulties similar to those encountered by deterministic LP builders using earlier computer languages. Because AMLs and other modelling systems do not yet support stochastic modelling, SLP formulations and solver links must often be custom programmed in one-off pieces of code, resulting in tight coupling among the model specification, model instance and the solver used. This can require extensive time and skill for programming and debugging, increasing the cost and effort required to use SLPs and limiting the practical realization of their analytical benefits.

The purpose of this paper is to help to advance development of algebraic modelling languages as SLP formulation and management tools by analyzing requirements and proposing supporting language extensions. We first review the capabilities of AMLs, their limitations for SLP formulation and management, and a solver input standard which could improve solver integration with AMLs. After introducing two sample problems, we present a taxonomy of SLP types to be supported, based on characteristics of their random structures which determine minimum model and data specifications. AML formulation techniques and solver integration mechanisms are then proposed and illustrated for most problem types, followed by a discussion of SLP model management needs and suggestions for further related research.

2 Background

In this section we summarize the process of model formulation, error checking and solver integration provided by AMLs, concluding with a brief discussion of their current and potential model management capabilities.

2.1 *Model formulation*

The process of using an AML system to formulate and solve an LP follows the steps shown in figure 1. First, a model is formulated using algebraic language constructs and data is specified (in either the model file or a distinct file) to instantiate the model. The AML system then generates output which is either directly readable by one or more solvers or is converted to solver input by a custom-written routine. A solver is then invoked through commands internal or external to the AML system and produces a problem solution.

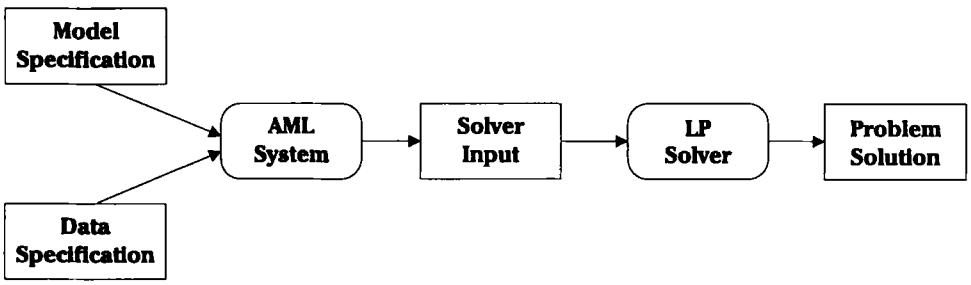

Figure 1. Steps in using an AML system.

In formulating models, AMLs use declarative statements (specifying what is to be done rather than how it is done), extensive use of domains over sets, and rows (constraints) rather than columns (activities) to represent models; they thus tend to support a mathematician's view of models rather than a view based on program-

ming language syntax or on activities and processes (Greenberg and Murphy [21]). Examples of algebraic modelling languages include GAMS (Brooke et al. [7]), AMPL (Fourer et al. [15]), LINGO (Cunningham and Schrage [10]), LPL (Hürlimann [24]), and MODLER (Greenberg [20]). For surveys of AML features and capabilities, see Greenberg and Murphy [21] and Kuip [32].

Algebraic modelling language systems allow modellers to distinguish between models and their instances in a way not done by earlier-generation matrix generator programs, which had to be custom-written for specific problems and typically mixed data and model specifications. As described in current object-oriented terminology, AML model specifications define models abstractly through object classes, including classes of indices, parameters and variables; general relationships, (defined through constraint classes); and class attributes such as nonnegativity. Model instances are formed when data sets fix membership in these classes. This approach has the advantage of producing a generic model specification for which many modifications can be done through changes in data sets alone.

These capabilities help modellers formulate a wide range of LPs and nonlinear programs, including those with complex time and network structures. However, stochastic LP formulation requires specification of random structures, ideally minimizing redundancy and problem size. In an earlier paper [17], we demonstrated SLP formulation and instantiation using one AML by explicitly constructing event tree indices and parent scenario references for the "deterministic equivalent problem", but the process is time consuming, cumbersome and error-prone. Use of AMLs for stochastic linear programming would be greatly facilitated by inclusion of specific language constructs and operations to automate the process by generating random variable realizations, dependent parameters, and the necessary indices, and carrying out automatic consistency checks.

2.2 *Consistency checking*

For deterministic LPs, the AML makes sure that the problem is set up correctly, that all index classes are consistent, that all data items are initialized, that attributes such as nonnegativity and integrality hold where specified, and that explicit modeller-defined consistency checks are satisfied. When the LP solver takes over, it still has to figure out the size of the problem, the location and number of nonzeroes, the starting basis, and so on.

For stochastic LPs, we expect the AML to perform similar functions: it should check that the problem is well-defined, recognize the time and stochastic structures, and pass this information on to the solver. Because there are many different SLP solution strategies, the output from the AML has to be quite flexible. Maximal flexibility is achieved when using the minimal problem representation, for which the existing S-MPS standard (see below) can act as a guideline, making suitable extensions where necessary.

2.3 *Solver integration*

Once models are formulated, AML systems use several approaches to communicating with solvers. A common solver integration approach is to transfer data through shared memory without the use of intermediate files. AMPL, for example, can communicate with MINOS, CPLEX or OSL in this fashion, giving fast and efficient model solution as a single operation after a model is formulated and instantiated. A second approach is to generate an output file in the standard MPS format read by essentially all current LP solvers; this allows transfer of formulated model instances among a wide range of solvers and facilitates solver comparison. Finally, AMLs may support solvers by producing output files that can be read into the solver directly or, if necessary, converted through custom-written routines to solver-specific input formats. This requires additional time and programming skill for creating the linking routines, but it avoids some of the shortcomings of the MPS format, such as limits to accuracy and variable names.

Each of these approaches presents some difficulties for stochastic modellers. Since SLP solvers are not currently supported by AML systems, integration must take place either through custom-written linking routines, which have the disadvantages outlined above, or through use of the MPS input format, which requires the use of the deterministic equivalent LP formulation and thus rules out some interesting SLP classes as well as placing size limitations on the problems that can be handled. Neither prospect provides SLP modellers with minimum-size model specifications capable of being readily passed to multiple solvers – a practice that is often desirable since SLP solver efficiency varies significantly with problem structure and is sometimes difficult to predict without experimentation.

2.4 *MPS format and the S-MPS extension*

Instances of deterministic LPs are most frequently stored and communicated using the well-known MPS standard [25,43]; this uses a 72-column card image with fixed line formats providing for up to three eight-character name fields and two twelve-digit numerical fields. Positions of numerical data are identified by matching column and row names. A model specification file consists of a number of sections, some of which are optional. Each section begins with a header line and most contain data lines; sections are briefly described in figure 2. MPS can be used to specify deterministic equivalents for SLPs which use only discrete distributions, but it is necessary to explicitly define all random variable realizations and resulting decision variables and parameters for all scenarios. Although this approach allows SLPs to be passed among solvers, its formulations are redundant wherever variables and parameters are not dependent on random variables, and it does present obvious practical limits on the sizes of problems that can be easily handled.

The S-MPS solver input standard defined by Birge et al. [4] extends the MPS standard to stochastic LPs in a nonredundant specification which is transferrable

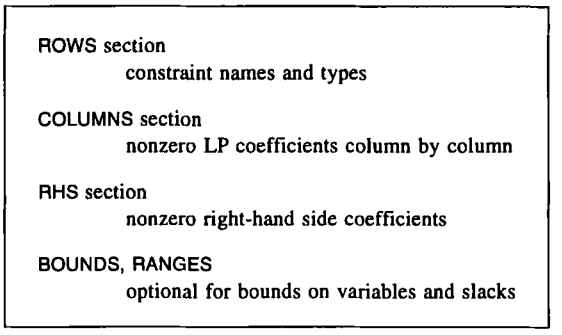

Figure 2. Summary of the MPS format.

among many SLP solvers now in use in research or application environments. S-MPS will also handle problems with continuous distributions. It consists of three files: the "Core" file, which closely follows the MPS format and contains the basic deterministic problem; the "Time" file, which explicitly defines the problem's time structure; and the "Stoch" file, which defines the random structure and specifies variables and parameters which are dependent on the problem's random variables. Each of the last two sections presents only the minimum data needed to extend the basic deterministic problem, so that problem size is controlled. The standard's explicit description of time and random structures also provides necessary input to decomposition SLP solvers. It thus provides output in a format which both minimizes redundancy and allows transfer among solvers without the custom routines now required by AML systems. However, the rigid record format imposes limitations which make multivariate random variables difficult to implement, as described later in the paper.

2.5 *AML model management capabilities*

We regard model management as encompassing support for the entire modelling life cycle, including problem identification, model creation, model implementation, validation, solution, interpretation, maintenance, and version and security control (cf. Krishnan [31]). Using this definition, AML systems as described above are limited in their model management functions to supporting formulation, consistency and completeness checking and solver links. However, management of optimization models is an active research area (e.g. see [45]), and AMLs can be accompanied by or embedded in additional software with extended model management capabilities. For example, AIMMS [6] provides interactive, Windows-based output display and report formatting; AMPL and GAMS both provide text-based report formatting; MODLER and its companion ANALYZE [19,20] give multiple model views and text explanations of model structure and output; and LPForm [35] formulates models in GAMS through either a graphic or text-based Windows interface.

Research on management of SLPs is relatively rare, although Wallace and Wets [48,49] consider preprocessing functions for detecting conditions for relatively complete recourse, and Kall and Mayer [27–29] have designed and partially implemented a model management system for SLPs which integrates GAMS with multiple SLP solvers. As far as we are aware, little consideration has so far been given to SLP management features within AML systems.

3 Two sample problems

This section introduces two small sample problems that will be used in the remainder of the paper. The original formulation of both problems is as deterministic LPs, but we will subsequently modify them to study alternative ways to treat stochastic information and possible algebraic modelling language problem formulations.

3.1 *Transportation problem*

The first problem is a completely standard transportation problem: units of a commodity have to be distributed from a number of warehouses or sources to a number of demand points or destinations. The allocation should minimize the total transportation cost while respecting supply constraints at each of the sources (one cannot ship more units than are available) and demand constraints at the destinations (demand must be satisfied).

The transportation model can be formulated mathematically as

$$\begin{aligned}
\text{minimize} \quad & \sum_{i,j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{i} x_{ij} \geq d_j, \qquad j = 1, \ldots, J, \\
& \sum_{j} x_{ij} \leq s_i, \qquad i = 1, \ldots, I, \\
& x_{ij} \geq 0, \qquad i = 1, \ldots, I, \quad j = 1, \ldots, J,
\end{aligned}$$

where c_{ij} represents the cost of transporting one unit of the commodity from source i to destination j, x_{ij} is the amount shipped, s_i is the amount of the commodity available for shipment from source i, and d_j is the demand of the commodity at destination j.

The same model can be formulated in the algebraic modelling language AMPL as follows:

```
set sources;
set destinations;

param supply {sources};
param demand {destinations};
param unit_cost  {sources, destinations};
```

```
var units_shipped {sources, destinations} >= 0;
minimize total_cost:
   sum {s in sources, d in destinations}
            unit_cost[s,d] * units_shipped[s,d];
subject to satisfy {d in destinations}
   sum {s in sources} units_shipped[s,d] >= demand[d];
subject to avail {s in sources}:
   sum {d in destinations} units_shipped[s,d] <= supply[s];
```

The accompanying AMPL data file for a specific instance with just two sources and three destinations is:

```
set sources      := Bergen      Oslo;
set destinations := Trondheim   Stavanger  Lillehammer;
param supply:   Bergen          Oslo :=
                  100           250;
param demand:  Trondheim        Stavanger  Lillehammer :=
                  150             100          80;
param unit_cost: Trondheim      Stavanger  Lillehammer :=
        Bergen     10             5            20
        Oslo       20            10            10        ;
```

This particular instance corresponds to a linear program which can also be rendered in the standard MPS format as follows:

```
NAME        Transportation problem
ROWS
 N   COST
 L   SUPBGN
 L   SUPOSL
 G   DEMTRD
 G   DEMSTV
 G   DEMLIL
COLUMNS
      TRBGNTRD    COST        10.0
      TRBGNTRD    DEMTRD       1.0     SUPBGN      1.0
      TRBGNSTV    COST         5.0
      TRBGNSTV    DEMSTV       1.0     SUPBGN      1.0
      TRBGNLIL    COST        20.0
      TRBGNLIL    DEMLIL       1.0     SUPBGN      1.0
      TROSLTRD    COST        20.0
      TROSLTRD    DEMTRD       1.0     SUPOSL      1.0
      TROSLSTV    COST        10.0
      TROSLSTV    DEMSTV       1.0     SUPOSL      1.0
      TROSLLIL    COST        10.0
      TROSLLIL    DEMLIL       1.0     SUPOSL      1.0
```

```
RHS
      RHS          DEMTRD       150.0
      RHS          DEMSTV       100.0
      RHS          DEMLIL        80.0
      RHS          SUPBGN       100.0
      RHS          SUPOSL       250.0
ENDATA
```

As mentioned, the problem as stated is a deterministic LP without time structure; we will eventually introduce random demands (with known distributions) to turn it into a stochastic LP.

3.2 *Production/inventory problem*

The second problem is a multiperiod production and inventory problem. In each period, sufficient quantities of several products must be produced (subject to some resource capacity constraints), in order to satisfy the demand for this product in this period. Excess production can be put into storage (at a cost proportional to the number of units stored) to augment production in a later period. The amount of storage space is limited. The AMPL model file describing this problem class is as follows:

```
set products;
param T > 0;
param start_inventory    {products};
param max_production     {products};
param production_cost    {products};
param holding_cost       {products};
param demand {products, 1..T};
param max_inventory;
var make {products, 1..T};
var hold  {products, 1..T};
minimize total_cost:
      sum {p in products, t in 1..T} ( production_cost[p]*make[p,t]
                                     + holding_cost[p]*hold[p,t] );
subject to balance {p in products, t in 1..T}:
      if (t=1 then start_inventory[p]
            else hold[p,t-1])
       + make[p,t] = demand[p,t] + hold[p,t];
subject to prod_cap {p in products, t in 1..T}:
      make[p,t] <= max_production[p];
subject to hold_cap {t in 1..T}:
      sum {p in products} hold[p,t] <= max_inventory;
```

The following AMPL data file describes a three-period instance with a single product:

```
param T      := 3;
set products := widgets;
param start_inventory   := widgets     0;
param max_production    := widgets     4;
param production_cost   := widgets     0;
param holding_cost      := widgets     1;
param max_inventory     := 4;
param demand: 1  2  3 :=
              1  4  4;
```

This instance can also be given in MPS format, as follows:

```
NAME        Production/inventory problem
ROWS
 N   COST
 E   DEMD1
 L   PCAP1
 L   HCAP1
 E   DEMD2
 L   PCAP2
 L   HCAP2
 E   DEMD3
 L   PCAP3
 L   HCAP3
COLUMNS
      MAKE1     DEMD1      1.0     PCAP1     1.0
      HOLD1     DEMD1     -1.0     HCAP1     1.0
      HOLD1     COST       1.0     DEMD2     1.0
      MAKE2     DEMD2      1.0     PCAP2     1.0
      HOLD2     DEMD2     -1.0     HCAP2     1.0
      HOLD2     COST       1.0     DEMD3     1.0
      MAKE3     DEMD3      1.0     PCAP3     1.0
      MAKE3     COST       1.0
RHS
      RHS       DEMD1      1.0
      RHS       DEMD2      4.0
      RHS       DEMD3      4.0
      RHS       PCAP1      4.0
      RHS       PCAP2      4.0
      RHS       PCAP3      4.0
      RHS       HCAP1      3.0
      RHS       HCAP2      3.0
      RHS       HCAP3      3.0
ENDATA
```

This problem has a definite time structure. We will use it to show possibilities (and limitations) in setting up multistage stochastic (dynamic) programming problems.

4 A taxonomy of stochastic LPs

SLPs present a major challenge for modelling systems because they can exhibit a variety of random structures which affect numbers and redundancy of scenarios, decision variables and parameters. In addition, random variable distribution characteristics and interdependencies greatly influence the degree to which problems must be specified in a "brute force" fashion. The following taxonomy of SLP types is intended as an initial attempt to classify and clearly describe SLP problem types so that support requirements for AMLs and other model management systems can be identified.

Generally speaking, a stochastic linear program is a linear program in which some of the data are stochastic. We will only consider the situation when probability distributions (possibly multivariate) are available for all the stochastic data items. This can also be viewed as linear programming *under risk*.

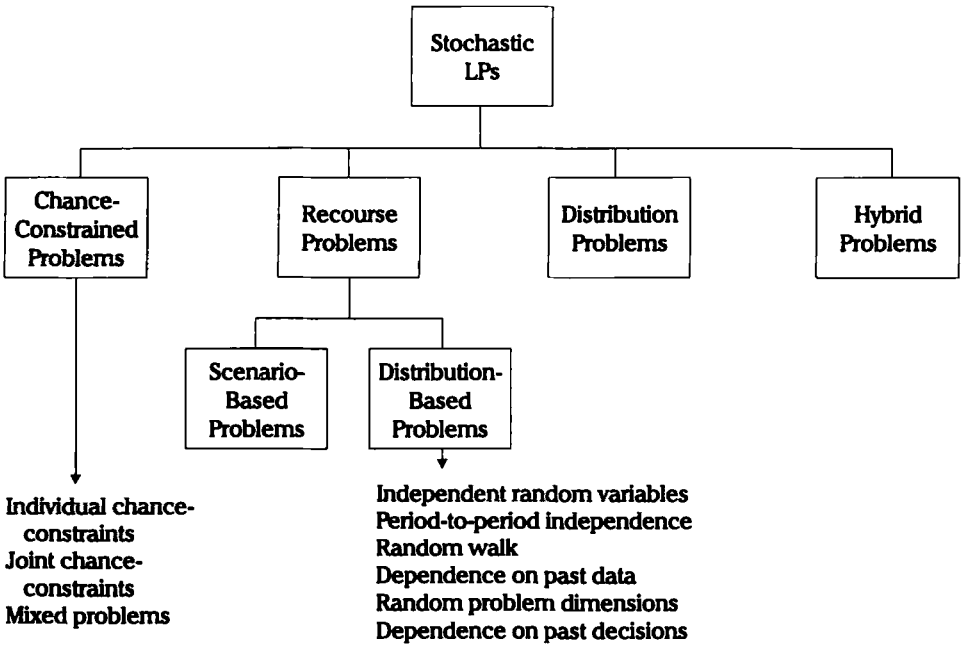

Figure 3. A taxonomy of stochastic LPs.

Figure 3 summarizes the taxonomy. We consider four main subclasses of SLPs. *Chance-constrained problems* (Charnes and Cooper [9], Prékopa [37], Kall [26]) contain one or more chance constraints or probabilistic constraints of the form

$$\text{Prob}\left[\sum_i a_{ij}x_j \geq b_j, j \in J_k\right] \geq \alpha_k,$$

where the coefficients a_{ij} and b_j are (possibly) random variables, J_k is some subset of the rows of the problem, α_k is a specified reliability level and the probability is taken with respect to the distribution of the random variables a_{ij} and b_j. The index k runs from 1 to K, the number of chance constraints contained in the problem. In problems with individual chance constraints, each of the index sets J_k is a singleton set; joint chance-constraint problems contain a single multivariate chance constraint, that is, $K = 1$. Mixed problems contain several multivariate chance constraints.

Recourse problems (Dantzig [12], Beale [1]) have the general form

$$\begin{array}{lllll}
\text{minimize} & c_0'x_0 + E_1\{c_1'x_1\} + & \cdots + E_T\{c_T'x_T\} & & \\
\text{subject to} & A_0x_0 & & & = b_0 \\
& B_1x_0 + A_1x_1 & & & = b_1 \\
& & \ddots & & \\
& & B_Tx_{T-1} + A_Tx_T & & = b_T \\
& x_t \geq 0, \ t = 0,\ldots,T. & & &
\end{array}$$

Any of the entries in the boldfaced matrices $\boldsymbol{A}_t$, $\boldsymbol{B}_t$ and vectors $\boldsymbol{b}_t$, $\boldsymbol{c}_t$ may be random. Time periods are denoted by $t = 0,\ldots,T$, and each period t has a decision vector $\boldsymbol{x}_t$ constrained explicitly by decisions in prior periods; E_t is used to denote the expected value of future decisions. The term "recourse" refers to the assumption that in any period one has access to a recourse decision at some penalty cost which will allow the constraints to be met.

The random structure of a recourse problem can be thought of as an (idealized) event tree (Lane and Hutchinson [33]) in which multiple realizations of random variables create branches at particular times and scenarios are paths from the event tree root to individual leaves. Each branch point in an event tree represents realizations of one or a combination of random variables at a given time period, given past realizations leading to the position in the tree from which the branch occurs. Associated with each node in a tree are the values of its random variable realizations, parameters calculated from those realizations, and contingent decision variables.

Scenario-based recourse problems have their tree structure, including random variable realizations and dependent data, explicitly specified. (In particular, this implies that all random elements must be finitely distributed.) These problems arise when it is convenient to explicitly describe event paths rather than state them in terms of distributions. Problems requiring a manager to give a subjective assessment of the outcomes of a range of possible future events are often of this type.

Problems for which *a priori* specification of distribution parameters can be used to generate the tree structure, decision variables and parameter values are referred to as *distribution-based problems*. Since SLP formulation issues are primarily concerned with event tree specification, we categorize distribution-based recourse problems according to the degree of distribution interdependence present in the random structure, which determines their formulation requirements as discussed in the following section. We distinguish the following problem types:

Type 1 (independent random variables): problems for which individual random variables are independent within and across time periods. For such problems the random structure can be completely specified using univariate distributions.

Type 2 (period-to-period independence): problems for which distributions are correlated within time periods but are independent of realizations in the past. Multistage versions of the transportation problem might well fall into this category if the transported good is a commodity such as umbrellas: the demand in the three demand centres will be weather-dependent and therefore correlated, while the demand in one year is unlikely to be affected by the sales in the previous year.

Type 3 (random-walk problems): those for which the distribution of the random variables is determined by the sum of an analogous random variable in the previous period and a random increment whose distribution does not depend on past events or decisions. As an example of this type, consider an investment problem which contains a simple interest rate model as one of its components.

Type 4 (dependence on past data): problems for which any distribution parameters (mean, discrete realizations, variance etc.) depend on past events. A problem in which interest rate changes follow different distributions when rates are high than when rates are low illustrates this type.

Type 5 (random problem dimensions): problems which allow the number of decision variables and/or constraints to be dependent on prior realizations. Problems which include "coffin states", in which branches can terminate before the end of the predefined time period, are a special case of this type. Examples of this include problems in which bankruptcy of a business might occur in one scenario, and ones in which new products might be introduced in particularly favorable circumstances.

Type 6 (dependence on past decisions): problems for which distribution parameters depend on past decisions (rather than just past events). Problems of this type occur in horse racing, when a single large bet can change betting odds, and stock markets, for a single large investor whose actions influence future prices and returns.

The third major category of SLPs are *distribution problems* [2]. Also called "wait-and-see problems", these problems are concerned with making statements about the distributions of objective value and decision variables obtained when solving a (deterministic) LP for each possible realization of the data.

These problem classes are not quite as exclusive as they may seem, and there are various possibilities for hybrid problems. One could envision, for instance, a setup in which part of the random structure is subjected to probabilistic constraints, while

explicit recourse is built in for the rest. Another problem of potential interest is a distribution problem containing one or two probabilistic constraints. We mention these problems mostly for the sake of completeness, as we are not aware of any serious attempts at using them in applications.

5 Formulation of chance-constrained problems

In this section we consider model and data specifications for chance-constrained problems, defining general randomness in the model specification but giving detail as data. Whenever necessary, we define new language keywords to support our formulations, for which examples are shown in AMPL. Hypothetical solver input in the S-MPS standard is given for each formulation to show output that could be produced by an AML system for automatic SLP solver integration.

5.1 *Individual chance constraints*

To define a problem with individual chance constraints, we must be able to specify in the algebraic modelling language the location of the random variables, along with their probability distributions. In addition, the language must be able to detect and process the individual chance constraints themselves.

The question arises which of these items should be part of the model, and which should be part of the data. It seems clear that the probability distributions are data items, since it cannot be desirable to change the model when substituting a uniform distribution for a truncated normal distribution, to give an example. On the other hand, the reliability levels associated with the constraints are a parameter class, and they should be treated like any other parameter class; that is, they should be set up in the model, with the data specified in the data section.

Similarly, the probabilistic constraints are a special type of constraint which should be set up as part of the model. This practically forces the point of view that the location of the random variables is also a model item, with the distribution being allowed to be a degenerate distribution if necessary. What one needs then is a construct in the modelling language to flag random variables.

Possible modifications to an AMPL model file, as applied to the transportation model, are as follows:

```
param reliability {destinations} >= 0, <= 1;
param demand {destinations} random;

subject to avail {d in destinations}:
   prob (sum {s in sources} units_shipped[s,d] >= demand[d])
                                                >= reliability[d];
```

The attribute random attaches to the parameter class demand in much the same way other attributes like integer, binary or >= 0 do. The keyword prob in the avail-

ability constraints indicates that a probability is to be evaluated. There is no requirement on the algebraic modelling language to actually "understand" this concept, but it is imperative that the information be passed on to the eventual solver in the correct way.

The actual distributions of the random demands would appear in the data file, as outlined in a separate section below in more detail.

The current version of the S-MPS standard does not include chance constraints, but individual chance constraints could be described as follows:

```
CHANCE
   G  RHS        DEMTRD      0.95
   G  RHS        DEMSTV      0.95
   G  RHS        DEMLIL      0.95
```

This would specify minimum reliability levels of 0.95 associated with each of the three demand locations.

5.2 *Joint chance constraint*

The modification from the individual chance constraints in the last section is a slight one, as seen in the following fragments of an AMPL file:

```
param reliability >= 0, <= 1;
param demand {destinations} random;
subject to avail:
    jointprob {d in destinations}
        (sum {s in sources} units_shipped[s,d] >= demand[d])
                                                >= reliability;
```

In place of an entire class of reliability levels, it is now sufficient to define a single parameter, but the probabilistic constraint has to take into consideration a multiple integral. This can be done by placing an indexing expression within the scope of the keyword prob; the use of a different keyword such as jointprob might be slightly more readable here.

In the S-MPS format, this could be rendered as

```
CHANCE        JOINT       0.98
      RHS     DEMTRD
      RHS     DEMSTV
      RHS     DEMLIL
```

but there is some ambiguity as to whether the coefficient of 0.98 forms a minimal reliability level or a maximal fault tolerance. The direction of the implied inequality cannot be conveyed within the framework of this standard.

5.3 *Specifying distributions*

Next we will take a look at how distributions can be specified within the data section.

Univariate distributions are relatively easy, but the language must be able to understand and deal with a number of widely used distributions such as uniform, normal, beta and gamma distributions, as well as ad-hoc distributions, both discrete and continuous.

As an example, we give part of an AMPL data file with proposed extensions:

```
param demand :=
    Trondheim    normal   (150,25);
    Stavanger    uniform  (80,120);
    Lillehammer  discrete (70 .25 80 .5 90 .25);
```

This describes stochastic demand in the transportation problem, where demand in Trondheim follows a normal distribution with mean 150 and variance 25, demand in Stavanger is uniformly distributed over the interval [80,120], and demand in Lillehammer follows the indicated three-point (discrete) distribution.

Other common univariate distributions can obviously be provided for in a very similar fashion. The only type that appears difficult is the general continuous distribution. This is, however, hardly surprising, since specification of such distributions requires definition of a nonlinear distribution (or density) function, for which the algebraic modelling languages, developed to handle linear programs, are not particularly well adapted.

The same distributions could be formulated in the S-MPS format:

```
INDEP          NORMAL
    RHS        DEMTRD      150.0          25.0
INDEP          UNIFORM
    RHS        DEMSTV       80.0         120.0
INDEP          DISCRETE
    RHS        DEMLIL       70.0           0.25
    RHS        DEMLIL       80.0           0.5
    RHS        DEMLIL       90.0           0.25
```

Due to the rigid card image format, two-parameter families of distributions can be handled easily, but additional parameters are rather difficult to describe without destroying the basic philosophy of the MPS format.

Multivariate distributions are somewhat harder to deal with because the information requirements are greater. For illustration purposes, we will assume that the stochastic demand in the three demand centres of the transportation problem is modelled by a trivariate normal distribution with mean $\mu = (150, 100, 80)$ and covariance matrix

$$\Sigma = \begin{pmatrix} 25 & 12 & 9 \\ 12 & 16 & 7.2 \\ 9 & 7.2 & 9 \end{pmatrix}.$$

(This describes an equicorrelated distribution with correlation coefficient 0.6 between the demands in any two destinations.)

The difficulty then is to communicate the dimension of the random vector and the locations of the individual entries. In AMPL we could use notation similar to the specification of two-dimensional tables, along the lines of

```
param demand := multinormal :
Trondheim     Stavanger    Lillehammer :=
mean          150          100          80
Trondheim      25           12           9
Stavanger      12           16           7.2
Lillehammer     9            7.2         9;
```

This device is not available in the S-MPS format because of the limitations posed by the rigid record structure. Instead, the original document [4] attempted to achieve the same result by decomposing the multivariate distribution into linear combinations of univariate ones. In the case of the multivariate normal distribution this is permissible since if x is a multivariate normal random variable with mean μ and covariance matrix Σ, then for any matrix T and vector a of appropriate dimensions, $Tx + a$ has a multivariate normal distribution with mean $T\mu + a$ and covariance matrix $T\Sigma T'$.

For instance, if $x^1 = (x_1^1, x_2^1, x_3^1)'$ is a random vector consisting of three independent standard normal random variables, $a^1 = (150, 100, 80)'$ and T^1 is defined as

$$T^1 = \begin{pmatrix} 5 & 0 & 0 \\ 2.4 & 3.2 & 0 \\ 1.8 & 0.9 & \sqrt{4.95} \end{pmatrix},$$

then the random variable $T^1x + a^1$ has the equicorrelated trivariate normal distribution given before. (T^1 is the Cholesky decomposition of the matrix Σ.)

This decomposition is not unique, however. For example, let us assume that x_1^2 has a (univariate) normal distribution with mean 150 and variance 25, x_2^2 is normally distributed with mean 35 and variance 16, and x_3^2 is normally distributed with mean 24.43974 and variance 9. If we define $a^2 = (0, 0, 0)'$ and

$$T^2 = \begin{pmatrix} 1 & 0 & 0 \\ 0.48 & 0.8 & 0 \\ 0.36 & 0.225 & \sqrt{0.55} \end{pmatrix},$$

then the random variable T^2x^2 has the same distribution as $T^1x^1 + a^1$.

Clearly, this decomposition is quite cumbersome, and the modeller should not be expected to perform the calculations by hand. One possible approach would be to have the decomposition calculated by the AML; alternatively, one might seek extensions to the S-MPS standard.

6 Recourse problem formulation

As noted above, recourse problems introduce randomization through explicit or implicit event tree structures which associate probabilities and expected values with contingent decisions during a planning period. Formulating these problems in the case of discrete distributions involves identifying random variables and either explicitly or implicitly stating the realizations that will form the underlying "tree" for the problem. Continuous distributions are handled slightly differently in that the discretizations are formed in the appropriate solver, either through random sampling or judicious approximations. For recourse problems as for chance-constraint problems, we separate as far as possible the randomization of parameters in the model specification from specification of distributions in a problem's data set.

6.1 *Scenario-based problem formulation*

The transportation model respecified as a two-stage scenario-based problem is shown below. In the model specification, a new index set scenarios has been introduced, with a probability parameter probab summing to 1 across all scenarios. Destination demand, along with associated shortage and surplus (recourse) costs are set up as random variables by indexing over destinations and scenarios. Decision variables are also indexed over scenarios, and the objective function minimizes expected total cost, as follows:

```
set sources;
set destinations;
set scenarios;

param supply {sources};
param demand {destinations, scenarios};
param unit_cost {sources, destinations};
param shortage_cost {destinations, scenarios};
param surplus_cost {destinations, scenarios};

param probab {scenarios} >= 0, <= 1;
        check: sum \I s in scenarios\r\ probab[s] = 1;

var units_shipped {sources, destinations} >= 0;
var units_short  {destinations, scenarios} >= 0;
var units_long   {destinations, scenarios} >= 0;

minimize total_cost:
    sum {s in sources, d in destinations}
            unit_cost[s,d]*units_shipped[s,d]
    + sum {d in destinations, w in scenarios}
                (probab[w]*shortage_cost[d]*units_short[d,w])
    + sum {d in destinations, w in scenarios}
                (probab[w]* surplus_cost[d]*units_long[d,w]);
```

```
subject to satisfy {d in destinations, w in scenarios}:
    sum {s in sources} units_shipped[s,d]
        + units_short[d,w] - units_long[d,w] = demand[d,w];
subject to avail {s in sources}:
    sum {d in destinations} units_shipped[s,d] <= supply[s];
```

For multistage problems of this type, it is necessary to keep explicit track of branches in the tree structure and parent scenarios in order to minimize redundancy in the specification. This is done by explicitly defining parent and start time parameters for each scenario, giving the number of the preceding scenario from which the current scenario branches and the time period in which the branch occurs. The tree structure is then explicitly stated in the problem's data set. The reader is referred to Gassmann and Ireland [17] for an example of a multistage scenario-based problem specification.

6.2 *Formulation of distribution-based recourse problems*

Distribution-based recourse problems offer the opportunity to let the AML system generate random structures automatically rather than rely on detailed tree specifications. As with chance-constrained problems, the modeller can change distributions more easily when the distribution is kept in the data specification and the system generates the random structure. Complexities arise, however, as distribution interdependencies become greater.

A generic two-stage distribution-based problem formulation in AMPL is shown below. Here the demand is defined as random and requires a distribution defined in the data set for each destination, as in the previous section. Decision variables units_short and units_long are defined as random to indicate that they will have multiple values corresponding to the demand distribution branches. The keyword expectation in the objective indicates that the system will calculate the expected value across all realizations of the demand random variable.

```
set sources;
set destinations;

param supply {sources};
param demand {destinations} random;
param shortage_cost {destinations};
param surplus_cost {destinations};
param unit_cost {sources, destinations};

var units_shipped {sources, destinations} >= 0;
var units_short {destinations} random >= 0;
var units_long {destinations} random >= 0;
```

```
minimize total_cost:
    sum {s in sources, d in destinations}
            unit_cost[s,d]*units_shipped[s,d]
    + expectation {d in destinations}
                    (shortage_cost[d]*units_short[d])
    + expectation {d in destinations}
                    (surplus_cost[d]*units_long[d]);
subject to satisfy {d in destinations}:
   sum {s in sources} units_shipped[s,d]
      + units_short[d,p] – units_long[d,p] = demand[d,p];
subject to avail {s in sources}:
   sum {d in destinations} units_shipped[s,d] <= supply[s];
```

This example serves to illustrate two-stage distribution-based models of both Type 1 and Type 2; Type 1 models will have independent demand distributions for the various destinations, while a Type 2 model would have dependent (multivariate) distributions as for the chance-constraint case discussed earlier.

Both the AMPL data file and the corresponding S-MPS file can take the form given in section 5.3. We illustrate here with independent discrete distributions as follows:

```
param demand :=
     Trondheim   discrete(135   .25   150   .5   165   .25);
     Stavanger   discrete( 90   .25   100   .5   110   .25);
     Lillehammer discrete( 70   .25    80   .5    90   .25);
```

and

```
INDEP          DISCRETE
     RHS       DEMTRD      135.0          0.25
     RHS       DEMTRD      150.0          0.5
     RHS       DEMTRD      165.0          0.25
*
     RHS       DEMSTV       90.0          0.25
     RHS       DEMSTV      100.0          0.5
     RHS       DEMSTV      110.0          0.25
*
     RHS       DEMLIL       70.0          0.25
     RHS       DEMLIL       80.0          0.5
     RHS       DEMLIL       90.0          0.25
```

This stoch file is sufficient to generate 27 different scenarios.The corresponding event tree can then be generated in the solver, relieving the AML of this task.

For multistage problems, the situation becomes more complicated. Information about the distributions alone is not sufficient to reconstruct the event tree; what is needed in addition is information about the time stage in which each random variable

is realized. This also means that the simple attribute random is inadequate. What is needed instead is a *hierarchy of uncertainties* corresponding to the time structure of the problem.

There are two ways to specify this in a language like AMPL, neither of which is completely satisfactory. In both cases, we start by defining the time structure using a special attribute or entity timeset, which must be ordered or capable of being ordered. It is then possible to define a *class* of attributes random[t] corresponding to the elements t of the time set. (For ease of use, one could allow the attribute random to be interpreted as equivalent to random[t2], where t2 is the second element of the time set, making it slightly more convenient to set up two-stage problems.)

It is possible to place this specification into the model file directly or, alternatively, to flag only general randomness in the model file and make more explicit statements in the data section. The first option might look as follows when applied to the production/inventory problem:

```
timeset periods := 1..T
param demand  {p in products, t in periods} random[t];
var   make    {p in products, t in periods} random[t];
var   hold    {p in products, t in periods} random[t];
```

The disadvantage of this approach is that a change in the distribution (a data item) may result in a change in the model. It is common practice, for instance, to set up multiperiod problems in two-stage stochastic form at first, in which all uncertainty will be resolved after the first period. Reformulating the problem as a true multistage problem would then necessitate changes in both the model and data file if this particular approach is followed.

If the explicit statement appears in the data file instead, then the model file for this example might look like

```
param demand  {p in products, t in periods} random;
var   make    {p in products, t in periods} random;
var   hold    {p in products, t in periods} random;
```

while the data file would be

```
param demand : widgets :=
     1     random[1]    1
     2     random[2]    discrete (3   .5    5    .5)
     3     random[3]    discrete (3   .5    5    .5);
```

Here, there are two problems. This type of specification is considerably more complicated to set up; it is therefore easier for the modeller to make a mistake. More importantly, variables do not appear in the data section, so the nature of the randomness of the variables make and hold is hard to specify. More work is clearly needed

here; we will put explicit information into the model file for the time being, giving the following full model file:

```
set products;
param T > 0;
timeset periods := 1..T;
param start_inventory   {products};
param max_production {products};
param production_cost {products};
param holding_cost     {products};
param max_inventory;
param demand {products, 1..T} random[t];
var make {products, t in periods} random[t];
var hold  {products, t in periods} random[t];
minimize total_cost:
     expectation(sum {p in products, t in periods}
          (production_cost[p]*make[p,t] + holding_cost[p]*hold[p,t]));
subject to balance {p in products, t in periods}:
     if (t=1 then start_inventory[p]
          else hold[p,t-1])
       + make[p,t] = demand[p,t] + hold[p,t];
subject to prod_cap {p in products, t in 1..T}:
  make[p,t] <= max_production[p];\
subject to hold_cap {t in 1..T}:
  sum {p in products} hold[p,t] <= max_inventory;
```

The corresponding data file looks like this:

```
param T      := 3;
set products := widgets;
param start_inventory    :=  widgets    0;
param max_production     :=  widgets    4;
param production_cost    :=  widgets    0;
param holding_cost       :=  widgets    1;
param max_inventory      := 4;
param demand: widgets :=
     1       1
     2       random(2, discrete(3   .5    5    .5))
     3       random(3, discrete(3   .5    5    .5));
```

Problem types beyond 2 must be multistage, since they all involve dependencies on prior random variable realizations. Type 3 problems can be set up by specifying parameter changes (rather than the original parameters) as random variables, with calculations in the model. This specification begins to break down the definition of distributions as data, however, since changing the demand distribution requires

modification of the demand calculation in the model itself. The separation can be maintained in a slightly different form by defining and storing the demand calculation and associated distribution in secondary model and data files, if the system allows this. In a system which permits multiple-model and data sections, such as AMPL, the production/inventory model could be rendered in random-walk form using a data file of this type:

```
param T      := 3;
set products := widgets;
param start_inventory  :=  widgets     0;
param max_production :=  widgets     4;
param production_cost :=  widgets     0;
param holding_cost     :=  widgets     1;
param max_inventory  := 4;
model;
param drift {products, t in 2..T} random[t];
subject to {p in products, t in 2..T}:
        demand[p,t] = demand[p,t-1] + drift[p,t];
data;
param drift: widgets  :=
     2     discrete(-1   .5    1    .5)
     3     discrete(-1   .5    1    .5);
```

Since the drift has a stationary distribution, it should be possible to simplify this further by specifying only one one-dimensional distribution:

```
model;
param drift {products, t in 2..T} random;
subject to {p in products, t in 2..T}:
        demand[p,t] = demand[p,t-1] + drift[p,t];
data;
param drift: widgets := discrete(-1    .5    1    .5);
```

Problems of Type 4 require more complex tracking of past realizations and their incorporation into formulas calculating distribution parameters. A sample suggested formulation is given below. As in Type 3, Type 4 distributions must rely on "model" specifications which could be kept in separate files from the original model definition to simplify distribution switching.

As an example, we give a modification of the production problem in which the step size of the random walk depends on the current demand level.

```
param T      := 3;
set products := widgets;
param start_inventory  :=  widgets     0;
```

```
param max_production := widgets    4;
param production_cost := widgets   0;
param holding_cost   := widgets    1;
param max_inventory  := 4;
model;
param drift {p in products, t in 2..T} random[t] :=
                        discrete (-0.1*demand[p,t-1]    0.5
                                  + 0.1*demand[p,t-1]   0.5);
subject to {p in products, t in 2..T}:
        demand[p,t] = demand[p,t-1] + drift[p,t];
```

Type 5 problems employ random problem dimensions. We can modify the production/inventory problem to illustrate some of the difficulties encountered with this problem type. Let us say that the first-period demand for widgets is known to be 1 and that second-period demand can be 2, 4, or 6 units, with probability 0.2, 0.5 and 0.3, respectively. If second-period demand is 2, we get out of the business altogether; there might be a lump-sum representing liquidation and salvage cost, but there will be no further production to be considered in the third stage. If second-stage demand is 4, we continue operations, with third-stage demand being 4, 5 or 6 units, with equal probability. Finally, if second-stage demand is 6, we will expand operation and storage facilities and introduce another product, skiffles. The two products are expected to be complementary, and we would estimate the joint demand distribution to look like

widgets	skiffles		
	4	6	8
6	0.25	0.06	0.00
8	0.06	0.25	0.06
10	0.01	0.06	0.25

The event tree for this problem takes the form given in figure 4, so it is possible to devise a scenario-based formulation of this particular instance. But what about a distribution-based formulation? First, the index set products must be allowed to be random. To guard against errors in the formulation, it makes sense to define a fixed

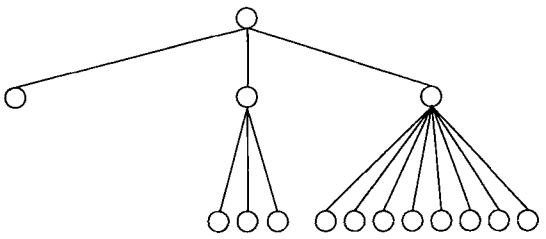

Figure 4. An event tree with a coffin state.

set potential_products and require that all realizations of the random set products (that is, all possible product mixes) be subsets of potential_products. This could be done in AMPL by specifying in the model file

```
set potential_products;
set products {1..T} within potential_products random;
# Actual product mix can depend on the observed data,
# but all possible products have been identified beforehand

param start_inventory {products[1]};
param production_cost {potential_products};
param holding_cost       {potential_products};
param demand {t in 1..T, products[t]} random;
param max_inventory;
param max_production {t in 1..T, products[t]};\

var make {t in 1..T, products[t]} random;
var hold {t in 1..T, products[t]} random;

minimize total_cost:
     sum {t in 1..T, p in products[t]}:
     (expectation(production_cost[p]*make[p,t] + holding_cost[p]*hold[p,t]);
```

In the data file we would have to specify something like

```
set potential_products := widgets skiffles;
set products :=
   1    widgets        # First-stage product mix is deterministic
   2    widgets        # Second-stage product mix is fixed
   3    if       demand[widgets,2] = 2 then { }        # Stop producing
        else if  demand[widgets,2] = 4 then widgets  # Continue as before
        else                                 widgets  skiffles;
# If second-stage demand is high, introduce another product

param demand :=
# First-stage demand is deterministic:
   1    widgets   1

# Second-stage demand is stochastic:
   2    widgets   discrete   2   .2   4   .5   6   .3

# Third-stage demand distribution depends on the current product mix:
   [3,*] if products[3] = widgets
         then widgets discrete (4   1/3   5   1/3   6   1/3)
         else if products[3] = widgets   skiffles
         then discrete(widgets:      6     8     10 :=
                       skiffles   4   0.25  0.06  0.01
                                  6   0.06  0.25  0.06
                                  8   0.00  0.06  0.25);
```

Type 6 problems (which we have not seen used in practical situations) pose the greatest difficulty and appear to require new solver approaches, since distributions are not determined until a problem is partially solved and the SLP cannot be solved with current techniques until the problem is fully known. Treatment of these clearly is an interesting area for further work.

6.3 *Summary of recommended AML system extensions*

As seen in the preceding examples of model and data files, the formulation of chance-constrained problems, scenario-based recourse problems and distribution-based recourse problems of Types 1 through 5, can be handled through relatively few language and processing extensions in an AML system with capabilities similar to AMPL. Recommended new language keywords include:

prob and jointprob to set up chance-constrained problems;

random and random[t] to identify random parameters, sets and variables;

expectation to express expectation operators in distribution-based recourse problems;

timeset to designate the set of time stages;

discrete, normal, uniform, etc. to specify particular distributions.

If S-MPS solver input were produced, AML system processing would have to be extended to generate its required files, but aside from the consistency checks described in the next section, relatively little new internal processing would be required beyond that since its field values are for the most part taken directly from the AML specifications. Difficulties arise in describing general (nonstandard) distributions and dependence of random variables across periods.

7 Model management functions

As noted earlier, management of SLP models is a promising and relatively new research area. We comment here on two model management functions which arise directly from our work on AML extensions: consistency checking for objective and constraint specifications, and automatic selection of the appropriate SLP solver based on AML model and data specifications.

7.1 *Consistency checks*

Algebraic modelling languages perform a number of consistency checks during the generation of the LP matrix. These checks include verification that all the indices referenced in objective and constraints are consistent with the definitions of parameters, sets and variables, that values are available for all referenced data items, and that attributes such as nonnegativity and integrality are satisfied.

In stochastic LPs, there are opportunities for additional checks which will prove useful to the modeller. In particular, constraints containing exactly one random parameter are ill-defined unless there is a probability expression that turns the constraint into a chance constraint. In the absence of a probability clause, there must be at least one additional random variable to balance things out.

For example, the demand constraint

```
transp[Bergen,Trondheim] + transp[Oslo,Trondheim] >= demand[Trondheim]
```

is ill-defined if demand is specified as a random variable and the level of transportation activity is set before this random variable is realized. Using the recourse formulation, the constraint can be salvaged by adding the induced random variables units_short and units_long and writing

```
transp[Bergen,Trondheim] + transp[Oslo,Trondheim]
  + units_short[Trondheim] – units_long[Trondheim] = demand[Trondheim]
```

For multistage problems, the consistency check will have to make use of the hierarchy of uncertainties described earlier. In essence, random variables of the highest level of uncertainty cannot occur singly in a constraint. In the three-stage production/inventory problem, for instance, the third-stage demand is not known when the second-stage production and inventory are set, both of which are random variables induced by the random demand in period 2. Hence, in the constraint

```
hold[p,2] + make[p,3] = demand[p,3] + hold[p,3]
```

make[p,3] and hold[p,3] must be defined as random[3] to balance the occurrence of the random variable demand[p,3].

7.2 *Solver selection*

A great number of different solution techniques are available and needed to solve SLPs, depending on problem type, characteristics of distributions within a problem, tree size and shape, and problem characteristics in general.

Problems with individual chance constraints have deterministic equivalents which are nonlinear programs, so in principle they could be solved by NLP solvers. Often, it is far more efficient, however, to use special techniques as described by Prékopa [37] and Szántai [46]. These methods can also be used to solve problems with joint chance constraints.

Two-stage recourse problems can be solved using stochastic quasigradient methods (Ermoliev [14] and Gaivoronski [18]), Benders decomposition (Van Slyke and Wets [47]), regularized decomposition (Ruszczyński [39,40]) or stochastic decomposition (Higle and Sen [22,23]). Multistage recourse problems can be solved by nested Benders decomposition (Birge [3], Gassmann [16]), scenario aggregation

(Rockafellar and Wets [38]), specialized interior point techniques (Lustig et al. [34], Birge and Holmes [5], Czyzyk et al. [11]) or similar methods. All of these methods can also be used iteratively within the approximation methods studied by Kall et al. [30].

To the extent that modelling experts can choose the appropriate solver based on characteristics of the model specification and data, AMLs should be able to support intelligent solver selection based on data about the nature of the SLP gathered during processing. This is clearly an area of future research, but we believe that it will be possible to formulate rules which would help the user decide between a stochastic decomposition approach and solving the deterministic equivalent by interior point methods, to give a very concrete example. Intelligent solver selection has been demonstrated for nonlinear programs by Schittkowski [41,42] and has been suggested for SLPs by Kall and Mayer [29].

8 Summary and conclusions

Based on a review of stochastic linear programming problem types and current algebraic modelling language capabilities, we have shown feasible, nonredundant formulation approaches for chance-constrained problems, scenario-based recourse problems and five types of distribution-based recourse problems. We have also shown that most of these problem types can be efficiently passed to SLP solvers using the flexible S-MPS solver input format and have identified difficulties with that format for some problem types. The formulations require a limited number of AML extensions and the implementation of consistency checks to ensure that constraints containing random variables are mathematically well-defined. In addition, data provided by these language extensions could support intelligent solver selection mechanisms; this and other SLP management functions such as multiple problem views and explanation facilities should provide many opportunities for further research.

Acknowledgements

This research was supported in part by funding from the National Sciences and Engineering Research Council of Canada (NSERC) and the Social Sciences and Humanities Research Council of Canada (SSHRC). The authors are grateful to two anonymous referees for carefully reading the manuscript and suggesting numerous improvements.

References

[1] E.M.L. Beale, On minimizing a convex function subject to linear inequalities, Journal of the Royal Statistical Society, Series B 17, 1955, 173–184.

[2] B. Bereanu, The distribution problem in stochastic linear programming, Operations Research Verfahren 8, 1970, 22–35.

[3] J.R. Birge, Decomposition and partitioning methods for multistage stochastic programs, Operations Research 33, 1985, 989–1007.

[4] J.R. Birge, M.A.H. Dempster, H.I. Gassmann, E.A. Gunn, A.J. King and S.W. Wallace, A standard input format for multiperiod stochastic linear programs, Committee on Algorithms Newsletter 17, 1988, 1–19.

[5] J.R. Birge and D. Holmes, Efficient solution of two-stage stochastic linear programs using interior point methods, Computational Optimization and Applications 1, 1992, 245–276.

[6] J. Bisschop and R. Entriken, *AIMMS: The Modelling System*, Paragon Decision Technology, 2001 DG Haarlem, The Netherlands, 1993.

[7] A. Brooke, D. Kendrick and A. Meeraus, *GAMS: A User Guide*, The Scientific Press, Redwood City, CA, 1988.

[8] D.R. Cariño, T. Kent, D.H. Myers, C. Stacy, M. Sylvanus, A.L. Turner, K. Watanabe and W.T. Ziemba, The Russell–Yasuda Kasai model: An asset/liability model for a Japanese insurance company using multistage stochastic programming, Interfaces 24, Jan.-Feb. 1994, 29–49.

[9] A. Charnes and W.W. Cooper, *Management Models and Industrial Applications of Linear Programming*, Vol. 1, Wiley, New York, 1961.

[10] K. Cunningham and L. Schrage, The LINGO modelling language, Technical Report, University of Chicago, Chicago, IL, 1989.

[11] J. Czyzyk, R. Fourer and S. Mehrotra, A study of the augmented system and column-splitting approaches for solving two-stage stochastic linear programs by interior-point methods, Technical Report 93-05, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, 1993.

[12] G.B. Dantzig, Linear programming under uncertainty, Management Science 1, 1955, 197–206.

[13] M.A.H. Dempster and A.M. Ireland, A financial expert decision support system, in *Mathematical Models for Decision Support*, G. Mitra, ed., NATO ASI Series F 48, Springer, Berlin, 1988, pp. 415–440.

[14] Yu. Ermoliev, Stochastic quasigradient methods, in *Numerical Techniques for Stochastic Optimization*, Yu. Ermoliev and R.J-B Wets, eds., Springer Series in Computational Mathematics, Vol. 10, Springer, Berlin, 1988, pp. 141–185.

[15] R. Fourer, D.M. Gay and B.W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, The Scientific Press, South San Francisco, CA, 1993.

[16] H.I. Gassmann, MSLiP: A computer code for the multistage stochastic linear programming problem, Mathematical Programming 47, 1990, 407–423.

[17] H.I. Gassmann and A.M. Ireland, Scenario formulation in an algebraic modelling language, Annals of Operations Research 59, 1995, 45–75.

[18] A. Gaivoronski, Implementation of stochastic quasigradient methods, in *Numerical Techniques for Stochastic Optimization*, Yu. Ermoliev and R.J-B Wets, eds., Springer Series in Computational Mathematics, Vol. 10, Springer, Berlin, 1988, pp. 313–351.

[19] H.J. Greenberg, *A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions*, Kluwer Academic, Boston, MA, 1993.

[20] H.J. Greenberg, *Modeling by Object-Driven Linear Elemental Relations: A User's Guide for MODLER*, Kluwer Academic, Boston, MA, 1993.

[21] H.J. Greenberg and F.H. Murphy, A comparison of mathematical programming modeling systems. Annals of Operations Research 38, 1992, 239–280.

[22] J.L. Higle and S. Sen, Stochastic decomposition: An algorithm for two-stage linear programs with recourse, Mathematics of Operations Research 16, 1991, 650–669.

[23] J.L. Higle and S. Sen, Guidelines for a computer implementation of stochastic decomposition algorithms, Technical Report, Department of Systems and Industrial Engineering, University of Arizona, Tucson, AZ, 1991.

[24] T. Hürlimann, Reference Manual for the LPL Modeling Language (Version 3.1), Institute for Automation and Operations Research, University of Fribourg, CH-1700 Fribourg, Switzerland, 1989.

[25] International Business Machines, Inc., *Mathematical Programming Subsystem – Extended (MPSX) and Generalized Upper Bounding (GUB) Program Description*, document number SH20-0968-1, 1972.

[26] P. Kall, *Stochastic Linear Programming*, Springer, Berlin, 1976.

[27] P. Kall and J. Mayer, SLP-IOR: A model management system for stochastic linear programming – system design, in *Optimization-Based Computer-Aided Modelling and Design*, A.J.M. Beulens and H.-J. Sebastian, eds., Springer, 1992, pp. 139–157.

[28] P. Kall and J. Mayer, A model management system for stochastic linear programming, in *System Modelling and Optimization*, P. Kall, ed., Springer, 1992, pp. 580–587.

[29] P. Kall and J. Mayer, Model management for stochastic linear programming, Working Paper, University of Zürich, Zürich, Switzerland, 1993.

[30] P. Kall, A. Ruszczyński and K. Frauendorfer, Approximation techniques in stochastic programming, in *Numerical Techniques for Stochastic Optimization*, Yu. Ermoliev and R.J-B Wets, eds., Springer Series in Computational Mathematics, Vol. 10, Springer, Berlin, 1988, pp. 33–64.

[31] R. Krishnan, Model management: Survey, future research directions and a bibliography, ORSA Computer Science Technical Section Newsletter 14, Spring 1993, 1–16.

[32] C.A.C. Kuip, Algebraic languages for mathematical programming, European Journal of Operational Research 67, 1993, 25–51.

[33] M. Lane and P. Hutchinson, A model for managing a certificate of deposit portfolio under uncertainty, in *Stochastic Programming*, M.A.H. Dempster, ed., Academic Press, London, 1980.

[34] I.J. Lustig, J.M. Mulvey and T.J. Carpenter, The formulation of stochastic programs for interior methods, Technical Report SOR 89-16, Department of Civil Engineering and Operations Research, Princeton University, Princeton, NJ, 1989.

[35] P. Ma, F.H. Murphy and E.A. Stohr, An implementation of LPFORM, Working Paper, University of Delaware, Newark, DE, 1994.

[36] J.M. Mulvey and H. Vladimirou, Stochastic network optimization models for investment planning, Annals of Operations Research 20, 1989, 187–217.

[37] A. Prékopa, Numerical solutions of probabilistic constrained programming problems, in *Numerical Techniques for Stochastic Optimization*, Yu. Ermoliev and R.J-B Wets, eds., Springer Series in Computational Mathematics, Vol. 10, Springer, Berlin, 1988, pp. 123–139.

[38] R.T. Rockafellar and R.J-B Wets, Scenarios and policy aggregation in optimization under uncertainty, Mathematics of Operations Research 16, 1991, 119–148.

[39] A. Ruszczyński, A regularized decomposition method for minimizing a sum of polyhedral functions, Mathematical Programming 35, 1986, 309–333.

[40] A. Ruszczyński, Parallel decomposition of multistage stochastic programs, Mathematical Programming 58, 1992, 201–228.

[41] K. Schittkowski, EMP: An expert system for mathematical programming, Technical Report, Mathematisches Institut, Universität Bayreuth, 1987.

[42] K. Schittkowski, Some experiments on heuristic code selection versus numerical performance in nonlinear programming, European Journal of Operational Research 65, 1993, 292–304.

[43] L. Schrage, *Linear, Integer and Quadratic Programming with LINDO*, 3rd ed., The Scientific Press, Palo Alto, CA, 1986.

[44] S. Sen, R.D. Doverspike and S. Cosares, Network planning with random demand, Working paper, Department of Systems and Industrial Engineering, University of Arizona, Tucson, AZ, 1992.

[45] B. Shetty, H.K. Bhargava and R. Krishnan (eds.), *Model Management in Operations Research*, Annals of Operations Research 38, 1992.

[46] T. Szántai, A computer code for solution of probabilistic-constrained stochastic programming problems, in *Numerical Techniques for Stochastic Optimization*, Yu. Ermoliev and R.J-B Wets, eds., Springer Series in Computational Mathematics, Vol. 10, Springer, Berlin, 1988, pp. 229–235.

[47] R. Van Slyke and R.J-B Wets, L-shaped linear programs with applications to optimal control theory and stochastic programming, SIAM Journal of Applied Mathematics 17, 1969, 638–663.
[48] S.W. Wallace and R.J-B Wets, Preprocessing in stochastic programming: the case of uncapacitated networks, ORSA Journal on Computing 1, 1989, 252–270.
[49] S.W. Wallace and R.J-B Wets, Preprocessing in stochastic programming: the case of linear programs, ORSA Journal on Computing 4, 1992, 45–59.