Forthcoming in *Applications of Stochastic Programming*, S.W. Wallace & W.T. Ziemba, eds. MPS-SIAM Series in Optimization, 2002.

# Stochastic Modelling and Optimization using STOCHASTICS™

M.A.H. Dempster, J.E. Scott & G.W.P. Thompson

Centre for Financial Research
Judge Institute of Management
University of Cambridge
&
Cambridge Systems Associates Limited
{mahd2, jes23, gwpt1}@cam.ac.uk
www-cfr.jims.cam.ac.uk

**Abstract**

Algebraic modelling languages have greatly simplified the formulation and management of deterministic mathematical programming problems, but as yet none of these languages provide any explicit support for the specification of *dynamic stochastic programming problems* (DSPs). Nevertheless, it is possible to describe a DSP with only the constructs available in deterministic languages using a so-called *nodal* formulation, and in this article we show how this can be done for a simple example problem. There are however, several situations when the nodal formulation becomes a limitation. Realistic stochastic programming problems may have very large deterministic equivalents, and if we can avoid it we do not wis h to instantiate these in full at any point during the modelling process. Also DSPs tend to have a highly repetitive structure, and it is worth going to some effort to exploit this for efficient problem generation. Finally the 'algorithm's form' of a stochastic programming problem is different to that of its deterministic equivalent, and the modelling system should take this into account. Addressing these points, we describe `stochgen`, the stochastic modelling component of the STOCHASTICS™ system, which works in conjunction with either AMPL or XPRESS-MP and allows the efficient generation of large-scale stochastic programming problems. We give some details of such problems and describe `solgen`, an implementation of nested Benders decomposition which works either independently of or in conjunction with `stochgen` and has been used to solve a variety of real world problems. We then discuss how visualization tools can be used to aid the DSP modelling process, and set out the progress we have made towards an integrated stochastic programming environment in the development of STOCHASTICS™.

**Keywords**: Dynamic stochastic programming, modelling languages, decomposition.

# 1 Dynamic stochastic programming

The canonical *two-stage dynamic stochastic programme* (DSP) with linear constraints can be written

$$\begin{array}{lllllll} \min\limits_{x_1} & f_1(x_1) & + & \mathbb{E}_{\boldsymbol{\omega}} \min\limits_{x_2(\boldsymbol{\omega})} f_2(x_2(\boldsymbol{\omega}), \boldsymbol{\omega}) & & & \\ \text{s.t.} & A_{11}x_1 & & & = & b_1 & \\ & A_{21}(\boldsymbol{\omega})x_1 & + & A_{22}(\boldsymbol{\omega})x_2(\boldsymbol{\omega}) & = & b_2(\boldsymbol{\omega}) & a.s. \\ & x_1 \geq 0, & & x_2(\boldsymbol{\omega}) \geq 0 & & & a.s. \end{array} \tag{1}$$

where $x_1 \in \mathbb{R}^{n_1}$, $x_2(\omega) \in \mathbb{R}^{n_2}$, $b_1 \in \mathbb{R}^{m_1}$, $b_2 \in \mathbb{R}^{m_2}$, $A_{11}$, $A_{21}(\omega)$ and $A_{22}(\omega)$ are appropriately dimensioned real matrices, the functions $f_1(\cdot)$ and $f_2(\cdot, \omega)$ are convex and *a.s.* indicates that the (in)equality holds *almost surely*, i.e. with probability one. We use $\boldsymbol{\omega}$ to denote a random vector variable associated with a probability space $\Omega$. The idea is to construct a linear programme where coefficients are known deterministically at present, but are uncertain (but drawn from some known distribution) at some time in the future. The term *dynamic* referring to (1) is used to indicate that in the applications we consider future uncertainty is represented by a continuous time vector stochastic process whose *data paths* $\omega$ will be approximated by a finite number of paths from a discrete time stochastic simulation, possibly with a *time step* much finer than the natural *time period* (e.g. a week or month) of the application. There may be several such time periods between *decision stages* of the model and thus $\Omega$ represents simulated states of the underlying data process sampled at the second stage date. (For more on this distinction between timescales in dynamic stochastic programming, see Dempster et al. (2000).)

Typically then, and for the purposes of this paper, we assume that the support of $\Omega$ is finite, i.e. that $\boldsymbol{\omega}$ can take only a finite number of values. In this case, we can consider solving the *deterministic equivalent* problem of (1) which, calculating the discrete expectation explicitly, can be written

$$\begin{array}{lllllll} \min\limits_{x_1, x_2(\omega)\ \omega \in \Omega} & f_1(x_1) & + & \sum\limits_{\omega \in \Omega} p(\omega) f_2(x_2(\omega), \omega) & & & \\ \text{s.t.} & A_{11}x_1 & & & = & b_1 & \\ & A_{21}(\omega)x_1 & + & A_{22}(\omega)x_2(\omega) & = & b_2(\omega) & \omega \in \Omega \\ & x_1 \geq 0 & & & & & \\ & x_2(\omega) \geq 0 & & & & & \omega \in \Omega, \end{array} \tag{2}$$

where $p(\omega)$ denotes the probability associated with the event $\omega$ under some probability measure $P$. The formulation in (2) is known as the *standard* (or *compact*) form deterministic equivalent. Here we have one vector variable $x_1$ which is "shared" between all realizations. Alternatively we can formulate the deterministic equivalent using a variable $x_1(\omega)$ for each $\omega \in \Omega$ and then add *nonanticipativity* constraints $x_1(\omega) = x_1(\omega'), \omega \neq \omega'$. This latter form is known as a *split-variable* formulation. It may be useful to make the nonanticipativity constraints explicit as their associated dual values carry sensitivity information about the structure of the event tree (Chen et al., 1998; Dempster, 1998; Dempster and Thompson, 1999). Also scenario decomposition-based solution methods start by relaxing these constraints (so that initially they solve $|\Omega|$ deterministic subproblems), and iterate towards a solution where the constraints are binding (Rockafellar, 1976; Dempster, 1988; Rockafellar and Wets, 1991; Ruszczyński, 1992).

The constraint matrix of (2) has a *dual block-angular* form corresponding to the problem's dynamic structure. As an alternative to solving the deterministic equivalent, we can exploit this structure using Benders decomposition (Van Slyke and Wets, 1969), a cutting plane method which splits the problem into a "master"

$$\min_{x_1} f_1(x_1) + \mathcal{Q}(x_1) \quad \text{s.t.} \quad A_{11}x_1 = b_1 \tag{3}$$

and a (separable) "slave" problem (often referred to as the *recourse* function)

$$\begin{array}{rcl} \mathcal{Q}(x_1) & = & \inf\limits_{x_2(\omega)} \sum\limits_{\omega\in\Omega} p(\omega) f_2(x_2(\omega), \omega) \\ & & \text{s.t.} \quad A_{22}(\omega)x_2(\omega) = b_2(\omega) - A_{21}(\omega)x_1 \quad \omega \in \Omega. \end{array} \tag{4}$$

It is easy enough to show that $\mathcal{Q}$ is a convex function of $x_1$. Benders decomposition proceeds by building up a piecewise linear lower approximation of $\mathcal{Q}$ by trying a sequence of different *proposals* $x_1^{(k)}$ for which dual solutions $\pi^{(k)}(\omega)$ are obtained. These are used to form *optimality cuts*, so that instead of solving the master problem directly we solve a sequence of problems of the form

$$\begin{array}{lrcl} \min\limits_{x_1} & f_1(x_1) + \theta & & \\ \text{s.t.} & A_{11}x_1 & = & b_1 \\ & \sum\limits_{\omega\in\Omega} p(\omega)\pi^{(k)}(\omega)A_{21}(\omega)x_1 + \theta & = & \sum\limits_{\omega\in\Omega} p(\omega)\pi^{(k)}(\omega)b_2(\omega) \quad k = 1, \ldots, K. \end{array} \tag{5}$$

If we have the case $\mathcal{Q}(x_1^{(k)}) = +\infty$ (i.e. for some proposals $x_1^{(k)}$ the recourse function is

infeasible), we can generate *feasibility cuts* by using directions of recession obtained from the (unbounded) dual solutions. For many practical problems Benders decomposition has been shown to be faster than solving the deterministic equivalent problem as usually only a few tens of cuts are required to accurately approximate $\mathcal{Q}$, and efficiencies can be gained because $\mathcal{Q}$ is in reality a collection of $|\Omega|$ similar small subproblems which can be solved very quickly when considered together. Also we can solve the subproblems of $\mathcal{Q}$ in parallel, and obtain very good speed-ups. Additionally the storage requirements of Benders decomposition codes are normally much smaller than general-purpose methods applied to the deterministic equivalent problem (2).

The ideas of the two stage formulation extend readily to the *multistage* case, where instead of progressing from no information to total information, information becomes available gradually as time progresses. To formally express this for a $T$-stage problem, we use a *filtration* $\mathcal{F} := \{\mathcal{F}_t : t = 1, \ldots, T\}$, where each $\mathcal{F}_t$ is a field with a generating partition $\mathcal{A}_t$ such that for all events $A \in \mathcal{A}_t$ there is $A' \in \mathcal{A}_{t-1}$ such that $A \subset A'$, that is, partitions $\mathcal{A}_t$ become progressively *finer*.

Rather than deal with filtrations explicitly, we generally think in terms of *event trees*, which have a node for each set in $\mathcal{A}_t$, $t = 1, \ldots, T$ and an arc between each node representing $A' \in \mathcal{A}_{t-1}$ and $A \in \mathcal{A}_t$. So, for example for the partition on the set $\Omega := \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5\}$:

$$\begin{aligned} \mathcal{A}_1 &= \{\Omega\} \\ \mathcal{A}_2 &= \{\{\omega_1, \omega_2\}, \{\omega_3\}, \{\omega_4, \omega_5\}\} \\ \mathcal{A}_3 &= \{\{\omega_1\}, \{\omega_2\}, \{\omega_3\}, \{\omega_4\}, \{\omega_5\}\}, \end{aligned} \tag{6}$$

we have the event tree shown in Figure 1. The usual tree terminology (ancestor, child, sibling, etc.) is used, and a single path from the root to one of the leaves is known as a *scenario*. Note that by assigning a probability to each leaf (or equivalently to each scenario), we unambiguously define the probability of all nodes in the event tree.

It is not strictly necessary that the partitions become finer at each period, and for some problems it may be appropriate to consider recombining event trees, particularly when the process to be modelled is discrete and has a small support. Archibald et al. (1999) describe a DSP where this is the case, however note that the decision process is always necessarily non-recombining, so the economy here is cosmetic rather than computational.

At each stage, we have a random variable $\boldsymbol{\omega}_t$, so that over time we have a stochastic process $\boldsymbol{\omega} := (\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_T)$ on the abstract probability space $(\Omega, \mathcal{F}, P)$. We use the nota-

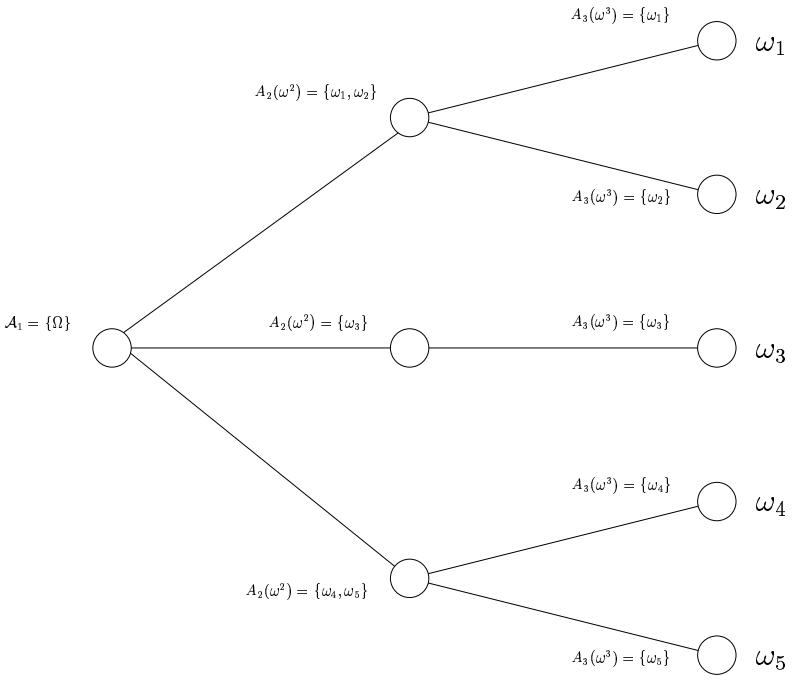


Figure 1: Event tree and partitions

tion $\boldsymbol{\omega}^t := (\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_t)$ to represent this process up to and including stage $t$. Similarly, we have a stochastic decision process $x(\boldsymbol{\omega}) := (x_1, x_2(\boldsymbol{\omega}_2), \ldots, x_T(\boldsymbol{\omega}_T))$ for which we use the notation $x^t(\boldsymbol{\omega}^t) := (x_1, \ldots, x_t(\boldsymbol{\omega}_t))$ to represent the process up to and including stage $t$. We can now express the linearly constrained *multistage dynamic stochastic programme* as

$$
\begin{array}{lllllllll}
\multicolumn{9}{l}{\min\limits_{x_1} c_1 x_1 + \mathbb{E}_{\boldsymbol{\omega}^2}\left\{ \min\limits_{x_2(\boldsymbol{\omega}_2)} f_2(x_2(\boldsymbol{\omega}_2), \boldsymbol{\omega}_2) + \mathbb{E}_{\boldsymbol{\omega}^3|\boldsymbol{\omega}^2}\left( \min\limits_{x_3(\boldsymbol{\omega}_3)} f_3(x_3(\boldsymbol{\omega}_3), \boldsymbol{\omega}_3) + \ldots \right.\right.} \\
\multicolumn{9}{r}{\left.\left. + \mathbb{E}_{\boldsymbol{\omega}^T|\boldsymbol{\omega}^{T-1}}\left[ \min\limits_{x_T(\boldsymbol{\omega}_T)} f_T(x_T(\boldsymbol{\omega}_T), \boldsymbol{\omega}_T)\right] \ldots \right)\right\}} \\
\text{s.t.} & A_{11}x_1 & & & & & & = & b_1 \\
& A_{21}(\boldsymbol{\omega}^2)x_1 & + & A_{22}(\boldsymbol{\omega}^2)x_2(\boldsymbol{\omega}^2) & & & & = & b_2(\boldsymbol{\omega}^2) \\
& A_{31}(\boldsymbol{\omega}^3)x_1 & + & A_{32}(\boldsymbol{\omega}^3)x_2(\boldsymbol{\omega}^2) & + & A_{33}(\boldsymbol{\omega}^3)x_3(\boldsymbol{\omega}^3) & & = & b_3(\boldsymbol{\omega}^3) \\
& & \vdots & & \vdots & & \ddots & \vdots & \\
& A_{T1}(\boldsymbol{\omega}^T)x_1 & + & A_{T2}(\boldsymbol{\omega}^T)x_2(\boldsymbol{\omega}^2) & + & \ldots + & A_{TT}(\boldsymbol{\omega}^T)x_T(\boldsymbol{\omega}^T) & = & b_3(\boldsymbol{\omega}^T) \\
& & & x_1 \geq 0 & & & & & \\
& & & x_t(\boldsymbol{\omega}^t) \geq 0 & & t = 2, \ldots, T, & & &
\end{array}
\tag{7}
$$

where again all constraints hold almost surely. Here $\mathbb{E}_{\boldsymbol{\omega}^t|\boldsymbol{\omega}^{t-1}}(\cdot)$ denotes conditional expectation with respect to the current state of the data process $\boldsymbol{\omega}$ conditioned on its past history. Again, this problem has both standard and split variable deterministic equivalent formulations (where variables in each stage $1, \ldots, T-1$ may be split) which we omit here for brevity. Both scenario decomposition and Benders decomposition based solution methods can be extended to efficiently handle the multistage case. The multistage Benders decomposition algorithm is known as *nested* Benders decomposition (Birge, 1985; Gassmann, 1990). The idea of the multistage formulation is that in this model each time an observation is made it is followed immediately by a decision, leading to the following sequence of observations and decisions:

$$
\begin{array}{ccccccccccc}
\text{decide} & & \text{observe} & & \text{decide} & & & & \text{observe} & & \text{decide} \\
x_1 & \rightsquigarrow & \omega_2 & \rightsquigarrow & x_2 & \rightsquigarrow & \ldots & \rightsquigarrow & \omega_T & \rightsquigarrow & x_T.
\end{array}
$$

In practice, because the set of events $\Omega$ is normally a discrete approximation to some real world continuous data process distribution, we model and solve a DSP such as (7) and implement *only* the optimal first stage decision $x_1$ since we cannot expect the real outcome of $\boldsymbol{\omega}^2$ to be close to any of the sampled realizations in the event tree. On the other hand,

this optimal first stage decision is robust against all future scenarios in $\Omega$ and in most applications analysis of the *a priori* optimal forward decisions provides a wealth of useful "what if?" information.

The problem for the modeller is to define at each node in the event tree the *coefficient process* $\xi_t(\boldsymbol{\omega}^t) := (A_{t1}(\boldsymbol{\omega}^t), \ldots, A_{tt}(\boldsymbol{\omega}^t), f_t(\cdot, \boldsymbol{\omega}^t), b_t(\boldsymbol{\omega}^t))$ in a representation suitable for solution either of the deterministic equivalent problem by a conventional deterministic solver, or in some representation suitable for solution by a DSP-specific solver (for example, the SMPS file format (Birge et al., 1987)). This is often referred to in the modelling language literature as the *algorithm's form*. As in the case of traditional deterministic mathematical programming, it is more convenient for the modeller to consider the problem in *modeller's form*, using set-theoretic and algebraic notation to represent the real-world situation to be modelled. In this paper, after introducing an example problem, we examine how AMPL, a deterministic algebraic modelling language, can be used to specify stochastic programming problems and produce either deterministic equivalent or DSP-specific algorithm's forms directly. However, realistic stochastic programmes are often very large, both in terms of the size of the event tree and the constraint dimensions $m_t$ and $n_t$, and typically they have a highly repetitive structure. In this case it becomes necessary to augment deterministic modelling languages with tools that can handle these structures, and in Sections 3-5 we describe the `stochgen` toolchain, which provides such facilities for the modelling languages AMPL and XPRESS-MP. In Section 6 we describe the STOCHASTICS™ `solgen` nested Benders solver and in Section 7 demonstrate its performance on a variety of real world large scale DSP problems. In the final section we describe our current work on STOCHASTICS™, in which we are developing a stochastic programming specific modelling language, as well as support tools to enable the visualization of problem and solution data for very large event trees.

# 2 An example problem

To provide a concrete example, we consider the modelling of a hypothetical portfolio management problem where the objective is to maximize the expected terminal value of a portfolio consisting of quantitites of a stock and a bond. At each time period $t = 1, \ldots, T$ the decision vector should tell us the optimal portfolio composition on each scenario. We have constraints to specifiy the initial portfolio value, and to disallow *short selling* (i.e. holding negative quantities of an asset). The first modelling task is to specify stochas-

tic processes for the two asset classes; a standard financial model is to assume that the (two-dimensional) price process $\boldsymbol{S}$ is a correlated geometric Brownian motion, i.e. each component $i$ of $\boldsymbol{S}$ follows the stochastic difference equation

$$d\boldsymbol{S}_i = \mu_i \boldsymbol{S}_i dt + \sigma_i \boldsymbol{S}_i d\boldsymbol{Z}_i, \tag{8}$$

where $\boldsymbol{Z}$ is a correlated Brownian motion. The *drift* ($\mu$) and *volatility* ($\sigma$) can be estimated from historical data. Of course the DSP framework assumes discrete time, so we model the movements of $\boldsymbol{S}$ with the stochastic difference equation

$$\frac{\boldsymbol{S}_{i(t+1)} - \boldsymbol{S}_{it}}{\boldsymbol{S}_{it}} = \mu_i \Delta t + \sigma_i \sqrt{\Delta t} \boldsymbol{\phi}_i, \tag{9}$$

where $\boldsymbol{\phi}$ is a correlated standard normal random vector. Given a covariance matrix $\Sigma$, a standard trick to generate values for $\phi$ is to generate uncorrelated normal deviates $\boldsymbol{\epsilon}$ and find a matrix $M$ such that $MM' = \Sigma$. Then $\boldsymbol{\phi} = M\boldsymbol{\epsilon}$, as from the definition of covariance $\Sigma = \mathbb{E}(\boldsymbol{\phi}\boldsymbol{\phi}') = M\mathbb{E}(\boldsymbol{\epsilon}\boldsymbol{\epsilon}')M' = MM'$.

Given the bivariate price process $\boldsymbol{S}$ as data process, the stochastic programme which we wish to solve is

$$\begin{array}{lrcll} & \multicolumn{4}{l}{\max\limits_{\substack{x_{it}(\boldsymbol{S}_t):t=1,\ldots,T,\\ i\in I}} \mathbb{E}_{\boldsymbol{S}^T}\left\{\sum\limits_{i\in I} \boldsymbol{S}_{iT} x_{iT}(\boldsymbol{S}_T)\right\}} \\ \text{s.t.} & \sum\limits_{i\in I} \boldsymbol{S}_{it} x_{i(t-1)}(\boldsymbol{S}^{t-1}) & = & \sum\limits_{i\in I} \boldsymbol{S}_{it} x_{it}(\boldsymbol{S}^t) & a.s. \quad t = 2,\ldots,T \\ & \sum\limits_{i\in I} \boldsymbol{S}_{i1} x_{i1} & \leq & 100 & \\ & x_{it}(\boldsymbol{S}^t) \geq 0 & & & a.s. \quad t = 1,\ldots,T \end{array} \tag{10}$$

where $I := \{\text{“stock”}, \text{“bond”}\}$ is the set of assets, and $x_{it}(\boldsymbol{S}^t)$ is the net asset value (NAV) held of asset $i$ at time $t$ given data history $\boldsymbol{S}^t$. The constraints here ensure that the total wealth is preserved between stages, that the initial cash requirement is less than or equal to \$100 and that the position is never shorted.

We should note here that this particular model is too simple to be realistic. No account is taken of the investor's attitude to risk—the portfolio which maximizes expected terminal wealth is also likely to have a high terminal variance. Also a realistic problem would have (as well as a much larger asset set) constraints on the change in portfolio composition between different periods and would take account of taxation and transaction costs. Without these considerations, it is possible to see that this problem is solvable analytically, in

particular, the myopic strategy that at each node of the event tree invests the entire wealth in the asset with the highest expected return in the next time step, i.e. that sets

$$x_{it}(\boldsymbol{S}^t) = \begin{cases} \sum_{j \in I} \boldsymbol{S}_{jt} x_{j(t-1)}(\boldsymbol{S}^{t-1}) & \text{if} \quad i = \arg\max_{j \in I} \mathbb{E}_{\boldsymbol{S}^{t+1}|\boldsymbol{S}^t} \left( r_{j(t+1)}(\boldsymbol{S}^{t+1}) \right) \\ 0 & \text{otherwise,} \end{cases}$$

where $r_{j(t+1)}(\boldsymbol{S}^{t+1}) := \left( \frac{\boldsymbol{S}_{j(t+1)} - \boldsymbol{S}_{jt}}{\boldsymbol{S}_{jt}} \right)$, is an optimal strategy. Also realistic models use more sophisticated price processes than geometric Brownian motion. Nevertheless the model is adequate for illustrative purposes and we will say how each of the mentioned extensions for realism can be handled in the course of this paper.

# 3 Representing event trees

For concreteness, we shall model the example problem with 3 stages using the partition specified in (6) and the probability measure $P(\omega_i) = 0.2$. Note that in reality a much larger event tree would be required to adequately discretize a data process such as the one given above.

We find that there are two convenient event tree representations. The first, most commonly used representation uses a *tree string*. This is a string of integers which specify for each stage the number of branches for each node in that stage. We normally write tree strings as a product of powers, so for example, the tree string $4.3.2.1^3$ generates a 7-stage event tree which has four branches in the first stage, three in each subtree of the second, etc. Obviously this allows only the specification of *balanced* trees (in the sense that each subtree in the same period has the same number of branches) but in the absence of more detail about the correct information structure this is normally adequate. For the specification of arbitrary event trees, we may use a *nodal partition* (NP) matrix (Consigli and Dempster, 1998b). This is a matrix with a row for each scenario and a column for each stage. We assign each node of the event tree a unique number, and the NP matrix entries $n_{kt}$ are node numbers, so that each row of the matrix shows which nodes in the event tree a scenario passes through. We say that an NP matrix is in *standard form* if $n_{ij} \leq n_{i'j}$ whenever $i \leq i'$ and $n_{ij} \leq n_{i'j'}$ whenever $j < j'$. Table 1 gives an appropriate nodal partition matrix for our example problem. Note that the nodal partition matrix is a redundant way of storing the tree, and it would be more efficient (for example) just to store the predecessor node of each node in the event tree. The NP-matrix representation however

| | $t=1$ | $t=2$ | $t=3$ |
|---|---|---|---|
| $k=1$ | 1 | 2 | 5 |
| $k=2$ | 1 | 2 | 6 |
| $k=3$ | 1 | 3 | 7 |
| $k=4$ | 1 | 4 | 8 |
| $k=5$ | 1 | 4 | 9 |

Table 1: Nodal partition matrix representation of event tree in Figure 1

has been maintained, primarily for historical reasons, and the overhead of storing it is not significant when compared to the storage requirement of the full stochastic programme. An alternative, similar representation for event trees is possible by considering the *scenario* partition, this is often referred to as the *Lane* matrix (after Lane and Hutchinson (1980)). An assumption in both of these representations is that only trees with a uniform depth are considered. Whilst this does not imply a loss of generality (as we can appropriately constrain decisions to be zero), it would not be the most efficient representation for event trees where this is not the case.

# 4 The nodal formulation

Having defined the event tree, we are in a position to instantiate simulator data over it. For the example problem, the price process we wish to simulate is sufficiently simple to be defined using the facilities available within AMPL. First we define the set of event tree nodes, a parameter to contain their probabilities and a parameter to contain their predecessors:

```
set nodes ordered;
param pred{nodes};
param prob{nodes};
```

We define the set of nodes as an "ordered" set (partially ordered by time) so that we can easily refer to the root and leaf nodes. To define the event tree of the example problem, we instantiate the above set and parameters with the following data block:

```
data;
param:  nodes:      pred prob :=
        1            .   1.0
        2            1   0.4
        3            1   0.2
        4            1   0.4
        5            2   0.5
        6            2   0.5
        7            3   1.0
        8            4   0.5
        9            4   0.5;
```

Here, the probabilities specified are the conditional probabilities of each node occurring given that its predecessor has occurred. In the case of a realistically large event tree, this data would normally be generated automatically, either from a nodal partition matrix or a tree string and read in from a file. The `stochgen` tool chain provides facilities for generating and manipulating tree structures and generating AMPL-compatible data files.

We also define the auxiliary objects `stage`, `stages`, `T` and `uprob` as

```
param stage{n in nodes} := if n = 1 then 1 else stage[pred[n]] + 1;
param T := stage[last(nodes)];
set stages := 1 .. T;
param uprob{n in nodes} := if n = 1 then 1.0 else uprob[pred[n]]*prob[n];
```

The parameter `stage` maps nodes to time stages, `T` is defined as the last decision stage and `stages` is the set of time stages. The parameter `uprob` is the *unconditional* probability of each node occurring.

To generate numeric data for asset prices, we define and assign data to the parameters $\mu$, $\sigma$ and $\Delta t$ from (9), and specify the correlation `c` between the two asset prices. We also require initial values for both assets. This is achieved by the following AMPL code:

```
set assets;

param mu   {assets};
param sigma{assets};
param c;
param dt;

param price1 {assets}; # price of assets at t=1
```

```
data;

param dt := 1.0;
param:  assets:    mu    sigma  price1 :=
        stock      0.15   0.2    50.0
        bond       0.10   0.1    50.0;

param c      := -0.3;
```

As in the case of the event tree, if we had a large number of assets we would store the data specification in a separate file. To generate correlated random deviates $\phi$ we first use the built-in AMPL function `Normal01` to assign standard (uncorrelated) normal random deviates to a parameter `e`, which is defined over each node in the event tree for each asset:

```
param e {n in nodes, i in assets} := Normal01();
```

This represents the statement

$$\boldsymbol{e}_{ti} \sim N(0,1) \qquad t = 1, \ldots, T \qquad i \in I. \tag{11}$$

Now, given the covariance matrix

$$\Sigma = \begin{pmatrix} 1 & c \\ c & 1 \end{pmatrix} \tag{12}$$

it is easy to verify that

$$M = \begin{pmatrix} \sqrt{1-c^2} & c \\ 0 & 1 \end{pmatrix} \tag{13}$$

is a suitable factorization. The AMPL code to perform the matrix multiplication $\phi = M\epsilon$ is then

```
param phi {n in nodes, i in assets} :=
        if i = "stock" then
          sqrt(1.0 - c*c)   * e[n, "stock"]
        + c                 * e[n, "bond"]
        else
          e[n, "bonds"];
```

At this point we have assigned correlated random deviates to each node of the event tree.

To specify the price process all that remains is to implement the difference equation (9) over the event tree as follows:

```
param price {n in nodes, i in assets} :=
        if n = first(nodes) then
          price1[i]
        else
          price[pred[n], i]) * (1 + mu[i]*dt + sigma[i]*sqrt(dt)*phi[n,i]);
```

Note that this definition is recursive (as are the definitions for `stage` and `uprob`), and hence relies on the fact that for any node `n`, `pred[n]` $<$ `n`.

In principle it is possible to express arbitrarily complicated stochastic data processes using an algebraic modelling language such as AMPL in this way, however in practice it may not be all that convenient. For example, if we have a large number of assets, we can obtain a value of $M$ by using Cholesky factorization of the covariance matrix $\Sigma$, however, AMPL does not provide a routine for this, and whilst it could be implemented using AMPL's imperative programming facilities, it would not be particularly readable or efficient. Furthermore, as we indicated earlier, difference equations for a realistic price process model (or for that matter any realistic stochastic process model) are normally substantially more complicated than (9) (see Wilkie (1987, 1995) for an example) and in this case, obtaining coefficients from historical data involves sophisticated econometric modelling, for which dedicated software (such as RATS or S-PLUS) provides a more appropriate environment. In the worst case (which is not particularly unusual) the data process simulator may only be available as a "black box" and we have no possibility of integrating it in an AMPL model. To provide an interface between an arbitrary simulator and a modelling language, the `stochgen` tool chain provides a program called `procgen`. Here we assume that the data process simulator can be encapsulated by the function

$$(s_t, s_{t+1}, \ldots, s_T) = f(k, s_{t-1}, s_{t-2}, \ldots, s_{t-l}). \tag{14}$$

That is, future states of a data process $s$ are a function of $l$ previous states and a seed $k$ for a random number generator. By allowing $l \neq 1$ we allow the possibility of non-Markovian data processes. It is possible to have $l > T$, but $l$ is fixed for all scenarios (the path simulator may choose to ignore initial conditions in some scenarios if they are unnecessary). We have $f$ produce $s_t, s_{t+1}, \ldots, s_T$ instead of just $s_{t+1}$ in order to avoid incurring any setup costs on the simulator more often than is necessary. `procgen` takes as input the function $f$ (it

is either called as an external program or is dynamically linked) and an event tree, and runs it once for each path in the event tree, in such a way that no state $s_t(\omega^t)$ is requested more than once, and the seed $k$ is updated in such a way that each generated path will be unique. The result is a set of nonredundant partial data paths in which the simulated data process realizations at nodes of the event tree with a common predecessor node have been generated conditional on the unique data path history to that node. Then numerical data for the entire event tree is output in a data format suitable for a variety of modelling languages or visualization tools. As noted above, we must also allow for the possibility that multiple simulator time steps are taken per time period, and that different stages in the same problem may have different numbers of periods. It is quite typical for example that the simulator produces monthly data, but decisions are required on a quarterly or annual basis and later stages contain several quarters or years.

Given this abstraction, the modeller need only define $f$ in a suitable way, we have found that people generally find this much simpler and less error-prone than implementing a "tree-aware" simulator on a case-by-case basis. There is an assumption here that the simulator is such that we do not need to know $s_t(\omega'_t)$ in order to generate $s_t(\omega_t)$, one can imagine models where this is not the case (for example in the generation of arbitrage free prices). In this situation, iteration over the tree is necessarily model-specific, but for the models we have encountered so far, the `procgen` abstraction has been sufficient.

We are now in a position to declare decision variables $x$ over the set of event tree nodes and for all assets as

```
var x{nodes, assets} >= 0;
```

This incorporates the no-shorting constraint; it is common modelling-language practice to put simple bounds such as this in the variable definition. We can define the objective function over the leaf nodes of the event tree, using the unconditional probability parameter `uprob`:

```
maximize expected_terminalwealth:
        sum{i in assets, n in nodes : stage[n] = T} uprob[n]*price[n,i]*x[n,i];
```

and we define constraints constraints similarly:

```
subject to self_financing{n in nodes : stage[n] <> 1}:
        sum{i in assets} price[n,i]*x[n,i]
                = sum{i in assets} price[n,i]*x[pred[n],i];

subject to budget:
        sum{i in assets} price[first(nodes),i]*x[first(nodes),i] <= 100;
```

This completes the nodal formulation of the example problem. In effect, we have used the modelling language to define the standard form deterministic equivalent linear programming problem corresponding to the stochastic programme we wished to model. In order to solve this, AMPL will construct this problem in full and send it to an LP solver that the user has provided. This has negative implications for efficiency. For most DSPs of interest, the deterministic equivalent form contains a great deal of redundancy, since the number of coefficients that are stochastic is small compared to the total number of coefficients at each node, and data for each node in the event tree tends to have a similar structure to that for other nodes, especially other nodes in the same stage. By constructing the deterministic equivalent, we also disregard structure information that may be exploited by a DSP solution algorithm such as nested Benders decomposition or the primal-dual interior point method.

# 5 `stochgen` formulation

In order to avoid creating the deterministic equivalent problem, and so that the modeller can avoid dealing with the recursive definitions necessary for a nodal formulation, `stochgen` requires that a problem is defined which is representative of *one scenario* of the dynamic stochastic programme, that is, the problem that would be obtained if random variables were made deterministic. Instead of indexing variables and constraints over the set of event tree nodes as above, the modeller indexes over time stages. For the example problem, this leads to the following AMPL code:

```
var x{stages, assets} >= 0;

maximize expected_terminalwealth:
        sum{i in assets} price[T,i]*x[T,i];

subject to self_financing{t in 2 .. T}:
        sum{i in assets} price[t,i]*x[t,i]
                = sum{i in assets} price[t,i]*x[t-1,i];

subject to budget:
        sum{i in assets} price[1,i]*x[1,i] <= 100;
```

We refer to this problem as the *core* problem (it is the same as the core problem of the SMPS standard). The inputs to `stochgen` are the core problem, a description of the event tree, and a `procgen`-compatible data path simulator as described above. An advantage of this approach is that a user may start with an existing deterministic model and convert it to a stochastic programme by simply *adding* information which describes the stochastic data process and model coefficients or functions.

Running AMPL on the above problem would produce a single scenario. We use the imperative programming features of AMPL to repeat this process for each scenario in the event tree, taking care that the data process parameters (in the example problem just the `price` parameter) point to the appropriate nodes in the event tree. How precisely this is done depends on the format of the stochastic data, `procgen`-generated data can be processed with standard headers which we supply to define objects `process` and `path`, so that the definition of the price parameter becomes

```
param price{t in stages, i in assets} := process[ord(i,assets),path[t]];
```

Normally AMPL sends its output directly to a solver, but in this case, no solution occurs until the last scenario has been processed. Instead, `stochgen` takes the output of AMPL and generates an SMPS representation of the DSP. In order to do this however, the model must be annotated with the dynamic structure of the problem. To do this we use the `suffix` notation of AMPL to assign stage information to each variable and constraint in the model. For the example problem, appropriate code would be

```
let {t in stages, i in assets} x[t,i].order := t;
let {t in 2..T} self_financing[t].order := t;
let budget.order := 1;
```

In terms of the linear programme corresponding to the core problem the effect of this is to impose a block lower-triangular form on the constraint matrix by permuting rows and columns so that they are ordered by stage. Figure 2 illustrates the effect of this annotation on the core problems output by AMPL. The left picture shows the constraint matrix for a 4-stage instance of the example problem with an arbitrary ordering of rows and columns (as is generated if the suffix information is not supplied). The right picture shows the problem with stage ordering imposed, so that it is in a block lower-triangular form. This specification of the problem's dynamic structure is essential in any DSP formulation technique, not only for generating an SMPS representation but also so that we can apply scenario decomposition or nested Benders solution methods.

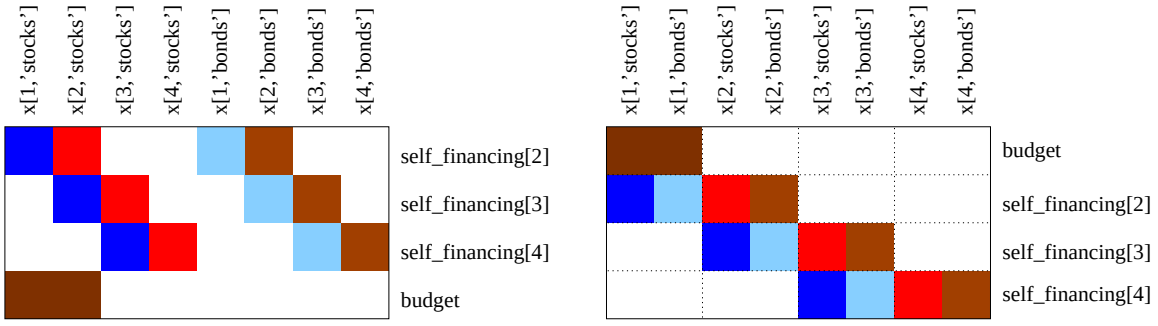


Figure 2: Constraint matrices of unordered and ordered core problems.

If at this stage we wish to generate the deterministic equivalent form, the `stochgen` toolchain provides `detgen`, which generates the deterministic equivalent in MPS form from the SMPS representation.

As well as simplifying the AMPL representation of DSP models, the `stochgen` formulation allows for more efficient generation when the event tree becomes very large, because the deterministic equivalent form is never stored in memory in its entirety, and `stochgen` is aware of the redundancies present in the DSP formulation. In Section 7 we will give details of problems which would have been impossible to generate using an AMPL nodal formulation in any reasonable amount of memory. Also for example in the context of importance

sampling (Infanger, 1994; Higle and Sen, 1996; Dempster, 1998) when it is necessary to regenerate the problem corresponding to only some part of the event tree, the `stochgen` framework has been used to do this efficiently, whereas by using a nodal formulation we would have been forced to regenerate the entire deterministic equivalent formulation. As far as we are aware, the `stochgen` component of STOCHASTICS™ is the only stochastic programming modelling system which has been used in this situation; other authors on the subject have had to "hard-code" their models.

# 6 Nested Benders decomposition using `solgen`

As noted above, the superiority of nested Benders decomposition over deterministic equivalent solution methods has been demonstrated repeatedly for a wide variety of problems. So far however there has been only one generally available implementation, called MSLiP, originally described in Gassmann (1990), which was extended in Dempster and Thompson (1998) to integrate it with commercial LP solvers and provide a parallel capability. Other two-stage Benders decomposition solvers are available (such as DECIS (Infanger, 1997) and IBM SP/OSL (King, 1994)) which can be adapted to solve multistage problems by using aggregation, but in our experience problems with a large number of stages generally benefit from a multistage implementation. Another possibility is to implement decomposition directly using the modelling language, the article by Gay and Gassmann in this volume gives details on how this might be achieved.

Part of the STOCHASTICS™ system is `solgen`, a new implementation of nested Benders decomposition which has been designed to be tightly integrated with modern modelling languages and LP solvers. The current version (1.30) of `solgen` uses CPLEX 7.1 to solve subproblems, and can read in problems in `MPS` or `SMPS` format, or those generated using AMPL, XPRESS-MP or the `stochgen` tools. Our emphasis has been to develop a solver which is both faster than currently available alternative methods and robust enough to be used in a general-purpose setting. In addition, the following features have been developed as part of our ongoing research:

- **Aggregation**. As was shown by Dempster and Thompson (1998), stage aggregation of the scenario tree can lead to accelerated solution times when using Benders decomposition. Our further research has shown that if the correct aggregation strategy is known, it is possible to get orders of magnitude speed-ups over the solution of the unaggregated problem or over the deterministic equivalent problem. If the modeller

is frequently solving similar problems, it is worth the effort to find these aggregations. An area of our current research is to find a heuristic model for choosing a good aggregation strategy *ab initio* and then use feedback from the solver as it runs to tune the strategy.

- **Regularization**. It has been known for some time that decomposition methods (both Benders decomposition and Dantzig-Wolfe decomposition) can behave poorly on some problems—the sequence of proposals generated by the standard algorithm can lead to a piecewise linearization of the recourse function which is as hard to handle as the recourse function itself. To avoid this situation we have investigated the use of regularizing terms on the objective function (following the work in Rockafellar (1976); Ruszczyński (1986); Ruszczyński and Świętanowski (1995)). The idea is to maintain an *incumbent* solution, choose proposals which are near it, and only change the incumbent when there is a demonstrable improvement in the linearization. We have used this method to successfully solve several problems which were previously either only amenable to deterministic equivalent methods, or insoluble, and to accelerate the solution of other problems. Currently the regularized method is only applicable to two-stage problems but a multistage implementation is in progress.

- **General convex objectives**. In financial applications it is important to be able to model general convex utility functions in order to handle investors' attitudes to risk and `solgen` takes two approaches to tackling such problems. The first method is to perform a further decomposition of the problem so that the convex objective is contained in an artificial 'final' stage which can then easily be solved (because it is unconstrained) and therefore approximated linearly. The second method is to use convex subproblem solvers to handle the convex objective directly. Currently the latter method uses CPLEX barrier as a subproblem solver, we hope to use a pivoting QP method soon and then employ SQP techniques to handle the general case.

- **Non-Markovian nested decomposition**. When the matrices $A_{ts}(\boldsymbol{\omega}^t), s < t-1$, in (7) are non-zero the problem is said to have a *non-Markovian* constraint structure. Such structures arise in multistage scheduling problems, and in financial problems which have complicated taxation or liquidity structures. A naïve approach is to introduce splitting variables to induce a Markovian structure, however this can result in a quadratic increase in the size of the problem. Instead, we have extended the nested Benders algorithm to handle non-Markovian problems directly, and modified `solgen`

appropriately. A number of problems have been solved using this new technique.

We have also been working on adaptations of the algorithm to solve certain non-convex problems which arise in portfolio management. These include the *fixed-mix* portfolio problem (which has a bilinear constraint on asset holdings) and problems which have *guaranteed return* or *Value-at-Risk* requirements, both of which can be modelled as probabilistic constraints.

# 7 A problem test set and computation times

In this section, we look at five problems which are drawn both from our own work and from the stochastic programming literature.

- **STORM**. A two stage stochastic freight scheduling problem (described in Mulvey and Ruszczyński (1995)) which is part of the POST standard problem set (available from `http://users.iems.nwu.edu/~jrbirge/html/dholmes/post.html`). Several other authors supply computation times for this problem, so we do so here for comparison.

- **WATSON**. An asset-liability management problem formulated by Dempster *et al.* for Watson Wyatt Worldwide, based on the CALM model (Dempster, 1993; Consigli and Dempster, 1998a) which is now made publically available (at `http://www-cfr.jims.cam.ac.uk/research/stprog.html`).

- **CORO**. A two-stage formulation of the HChLOUSO hydrocarbon logistics planning problem (Escudero et al., 1999). The problem here is from the "Case 3" dataset which involves planning the movement and spot market transactions of seven oil products between 41 ports, sales locations, and storage depots under uncertain local demands and prices.

- **DROP**. An alternative (more tightly constrained) formulation of the HChLOUSO problem (Dempster et al., 2000), also involving refining and detailed in this volume.

- **PFEX**. Here we have taken the example problem described above for two assets, and extended it so that it models 9 asset classes (stocks and bonds) in three currency regions. Interest rates for the bond processes use a mean-reverting model. In order

to make the problem realistic, a piecewise linear objective is used in order to give an attitude towards risk, and a liquidity constraint of the form

$$|S_{it}(\omega^t)x_{it}(\omega^t) - S_{it}(\omega^t)x_{i(t-1)}(\omega^{t-1})| \leq \nu \sum_{j \in I} S_{jt}(\omega^t)x_{jt}(\omega^t) \quad i \in I \quad t = 2, \ldots, T$$

is also imposed which stops changes in position between stages being more than a fraction $\nu$ of total wealth.

All problems (apart from **STORM**) were formulated using the `stochgen` modelling system in conjunction with either AMPL or XPRESS-MP. Table 2 gives the numbers of scenarios, stages and (standard form) deterministic equivalent problem dimensions for each problem.

| Problem | Stages | Scenarios | Rows | Columns | Non-zeros | Objective |
|---|---|---|---|---|---|---|
| **STORMG2.8** | 2 | 8 | 4394 | 10193 | 27424 | 15535235.73 |
| **STORMG2.27** | 2 | 27 | 14388 | 34114 | 90903 | 15508982.32 |
| **STORMG2.125** | 2 | 125 | 65936 | 157496 | 418321 | 15512091.18 |
| **STORMG2.1024** | 2 | 1024 | 526186 | 1259121 | 3341696 | 15802590.35 |
| **WATSON.10.256.C** | 10 | 256 | 43518 | 82177 | 218888 | 1849.40 |
| **WATSON.10.512.C** | 10 | 512 | 67070 | 128001 | 350728 | 1797.30 |
| **WATSON.10.1024.C** | 10 | 1024 | 134128 | 255987 | 701428 | 1798.42 |
| **WATSON.10.1920.C** | 10 | 1920 | 251442 | 479905 | 1315028 | 1778.36 |
| **WATSON.10.2688.C** | 10 | 2688 | 352014 | 671861 | 1841028 | 1687.72 |
| **CORO.2.10** | 2 | 10 | 155246 | 545633 | 1456120 | 24455.41 |
| **CORO.2.50** | 2 | 50 | 770446 | 2688633 | 7245640 | 24437.31 |
| **DROP.2.10** | 2 | 10 | 155271 | 500820 | 2182070 | 1179057.97 |
| **DROP.2.50** | 2 | 50 | 766471 | 2464820 | 10769670 | 1179083.72 |
| **PFEX.6.3840** | 6 | 3840 | 207043 | 109290 | 763379 | 14905.98 |
| **PFEX.6.7680** | 6 | 7680 | 412483 | 217770 | 1520819 | 7582.11 |
| **PFEX.6.30720** | 6 | 30720 | 1126723 | 609450 | 4147379 | 3019.06 |

Table 2: Test set problem dimensions

We give these statistics with the usual proviso that a problem's size gives only a very rough indication of how difficult it is to solve. In particular, the frequent emphasis on

producing problems with a very large number of scenarios is not always justified. We have observed (for example for portfolio management problems) that the solution stabilizes with quite modest numbers of scenarios providing the constraint structure acts to prevent myopic strategies such as that discussed in Section 1 from being optimal. Nevertheless these problems can be harder to solve than much larger problems which enjoy more separability. Although this point is obvious there has been a tendency in the computational literature on stochastic programming to solve problems with huge numbers of scenarios with little evidence that the formulation is not unrealistically underconstrained.

Table 3 gives solution statistics for the test set solved using `solgen` and (where available) for the same problems solved using the fastest of the CPLEX primal and dual simplex and barrier methods (indicated by P, D or B respectively in the final column). All experiments were run on an AMD Athlon 650MHz with 512MB RAM. Note that the solution times for all but the two smallest **STORM** problems are shorter for `solgen` than for any of the CPLEX algorithms, sometimes by an order of magnitude. Also, because `solgen` is aware of the redundancy in the DSP formulation, memory requirements are reduced and with the same machine much larger problems can be solved. For the **DROP** problems it was necessary to use a regularized master objective; currently this requires the master problems to be solved using an interior point method. Because interior point methods cannot easily be hot-started solution times are very long, but we are presently working on simplex-based methods which should be much faster.

# 8 Future developments

## `stochgen` 3

In this concluding section we document current and planned development of the STOCHASTICS™ system. The version of `stochgen` currently under development (shown in Figure 3) aims to provide the modelling, solution and visualisation features required by industrial DSP applications with a set of components and standalone tools controlled through a separate graphical user interface shown in Figure 3. Here we are using Excel for this, allowing simulator, model and solver parameters to be managed easily. The model (shown being edited on the left) is an AMPL nodal formulation of a portfolio management problem (a version of the CALM model described in Consigli and Dempster (1998a,b)), while the description of the event tree and simulator parameters are maintained separately in an Excel spreadsheet. Visualization of scenario data and the solution are provided by separate

| Problem | solgen time (s) | solgen memory (MB) | CPLEX time (s) | CPLEX memory (MB) | method |
|---|---|---|---|---|---|
| **STORMG2.8** | 2.41 | 39 | 1.30 | 6 | D |
| **STORMG2.27** | 8.15 | 46 | 7.04 | 16 | D |
| **STORMG2.125** | 45.04 | 63 | 89.25 | 70 | D |
| **STORMG2.1024** | 350.19 | 238 | 1363.58 | 688 | B |
| **WATSON.10.256.C** | 11.16 | 16 | 25.01 | 35 | B |
| **WATSON.10.512.C** | 12.92 | 21 | 46.60 | 55 | B |
| **WATSON.10.1024.C** | 32.80 | 39 | 116.61 | 110 | B |
| **WATSON.10.1920.C** | 55.46 | 78 | 207.95 | 205 | B |
| **WATSON.10.2688.C** | 92.93 | 100 | 323.74 | 287 | B |
| **CORO.2.10** | 269.66 | 19 | 1981.35 | 160 | P |
| **CORO.2.50** | 1003.11 | 38 | 22584.52 | 801 | P |
| **DROP.2.10** | 1688.49 | 36 | 2231.00 | 175 | P |
| **DROP.2.50** | 5623.66 | 60 | 52250.13 | 861 | P |
| **PFEX.6.3840** | 201.47 | 72 | 924.25 | 142 | B |
| **PFEX.6.7680** | 461.92 | 146 | - | - | - |
| **PFEX.6.30720** | 2228.40 | 435 | - | - | - |

Table 3: Test set computational results — `solgen` and CPLEX

Java components. Controlling the whole process from Excel allows automatic solve-resolve scripts (such as are needed to generate backtest results from historical data) to be written in VBA. All the `stochgen` 3 components can be accessed from other languages (Java, C, C++), if this is preferred to VBA, or even used as standalone tools.

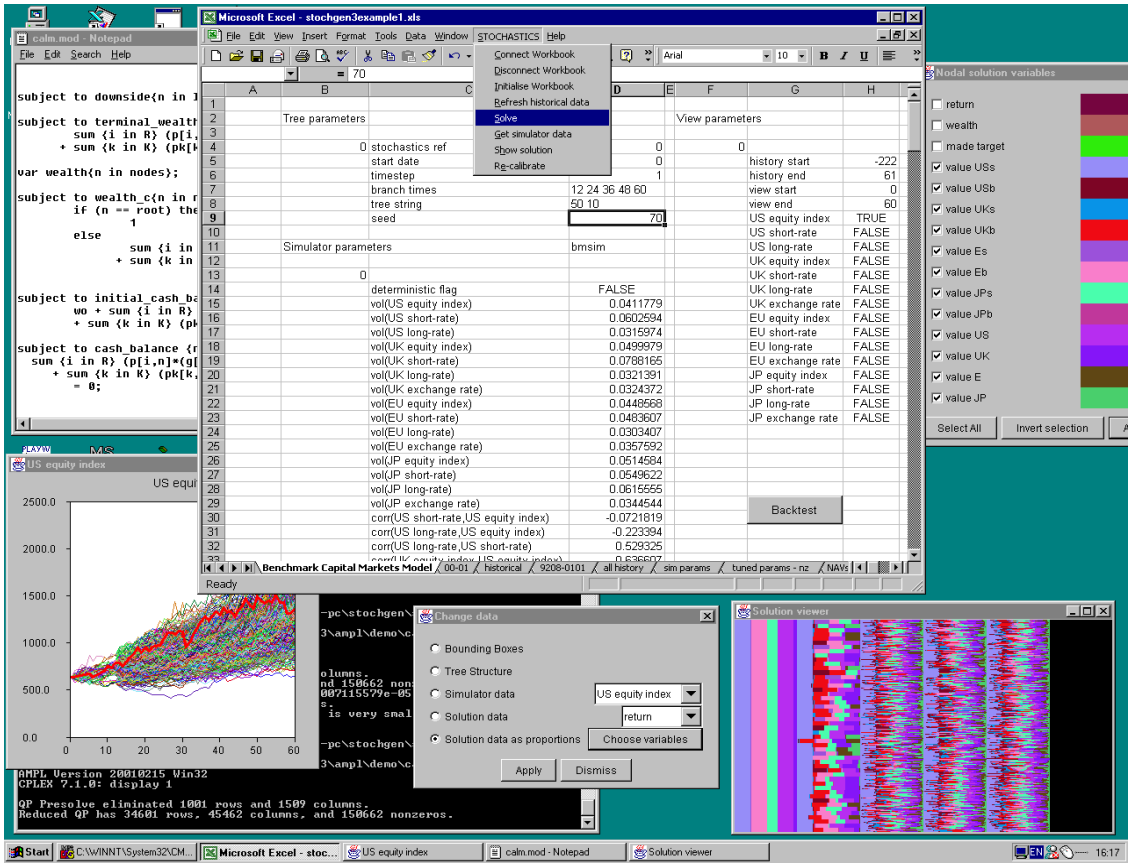


Figure 3: Using `stochgen` 3 to solve a portfolio management problem

## Visualisation

Once a stochastic programme has been solved, there remains the problem of viewing and analysing the solution. This forms a critical part of the model-solve-analyse cycle, which must be repeated several times before a satisfactory solution to a DSP problem is found. Fast and flexible visualization tools are thus essential if DSP is to be widely applied commercially.

DSP solution data consists of the value of the decision variables at nodes on a event tree, together with constraint dual information. While the variables along each scenario in the tree can be extracted and viewed using a variety of visualization tools (Excel, MATLAB, etc., or the component shown in Figure 3), these 'scenario-viewers' have the restriction

that the user can only easily see data along a few scenarios at once. Though this is often acceptable for visualizing paths of stochastic processes (see Figure 4 for example) the approach is less well suited to examining solutions of stochastic programs where one may have hundreds of thousands of scenarios which are subtly related to each other through the branching of the event tree.

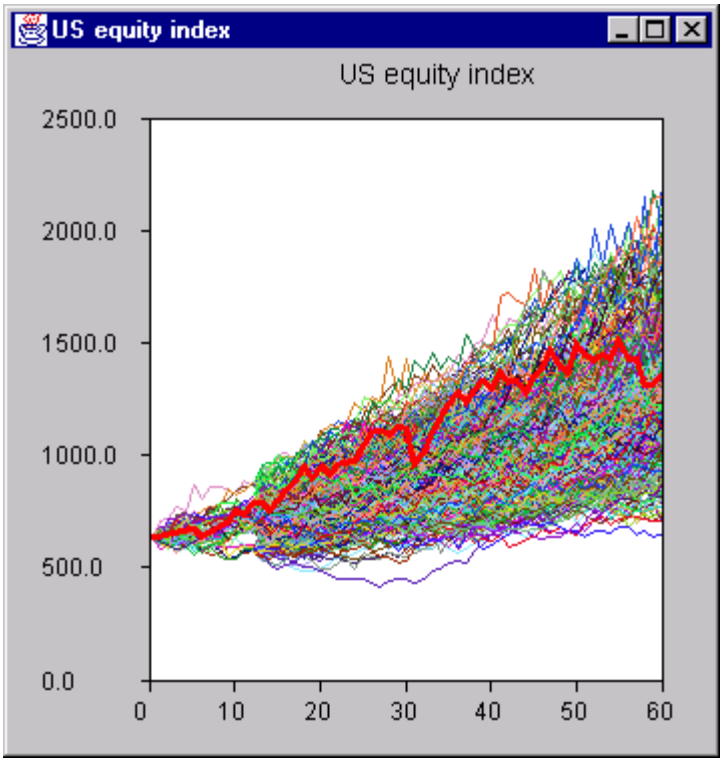


Figure 4: Scenario-based visualization

For example, given a collection of different scenarios with a similar sequence of decisions, we cannot say if the decisions are the same because the scenarios have not reached the point in the event tree where they branch from one another, or whether the solution happens to be rather non-stochastic in this region of the tree. With a scenario-based view, it becomes impossible to view data for all scenarios at once without seeing a dense 'cloud' of scenarios (as exemplified by Figure 4) in which this branching is obscured. The example in Figure 4 has only 500 scenarios, but it is already hard to distinguish one scenario from another. With 100,000 scenarios this problem would be insurmountable.

As part of `stochgen` 3, a 'tree viewer' component has been written which can be used to visualize and explore data defined on trees. This allocates a single rectangle for each node in the tree (Figure 5 shows the layout of rectangles for a tree with 2-2-2-2-2 branching) by placing nodes in the same time-period in a vertical column, with each node to the right of its predecessor node. The vertical ordering of nodes can also be chosen based on some function of the nodal problem or solution data.

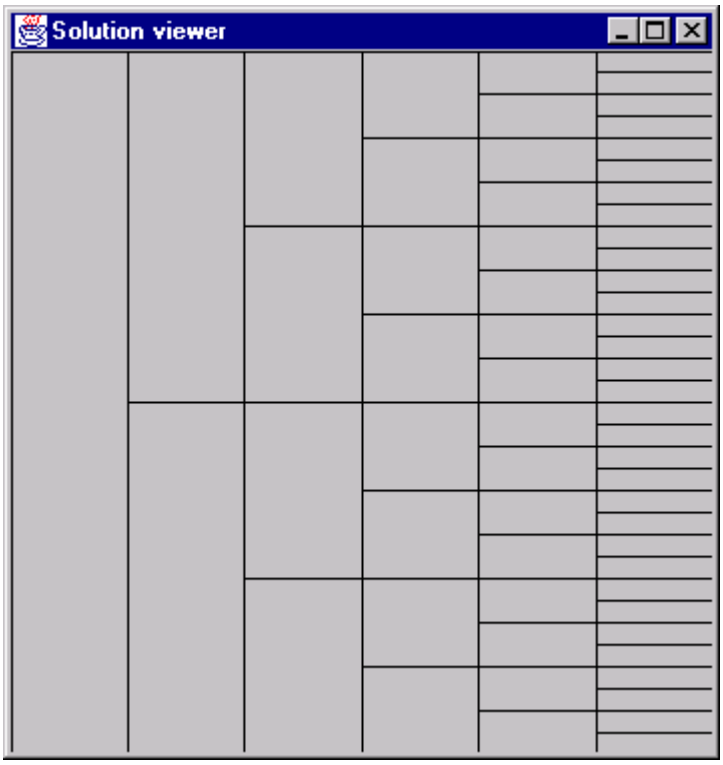


Figure 5: Tree-visualization, allocating a rectangle for each node, with nodes in the same time-period stacked vertically

The different rectangles can then be used to display solution information. The benefit of the tree-based view is that it allows us to see properties such as stability of solution over the tree, or to find regions of the tree in which the solution behaves unexpectedly. In Figure 6 we show the solution to the portfolio-management problem of Figure 3 by dividing each node's rectangle into vertical bars with width proportional to the proportion of the portfolio invested in each asset. (Note that nodes in the final stage do not have decision variables in this model and hence the last column is blank) Figure 7 shows a zoom-in of

part of the solution.

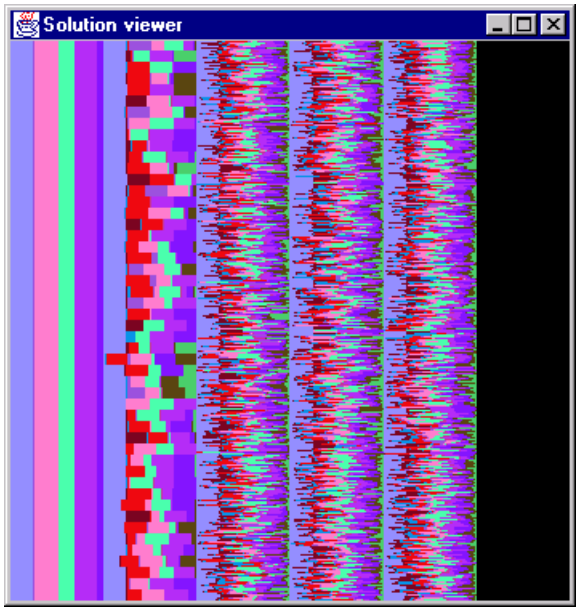


Figure 6: Tree-based visualization the solution to a portfolio management problem. Here the branching in the tree is 50-10-1-1-1.

## Problem generation

The main stumbling block to the solution of very large DSP's is to effect the generation of individual sub matrices invoking the stochastic simulator only when required by the solver. Such an approach is also essential for resampling techniques (Dempster, 1998; Dempster and Thompson, 1999), where we may deliberately remove part of the event tree if the resampling algorithm decides that its effect on the solution is minimal and instead generate a more 'bushy' tree in regions where the solution is sensitive to the discretization.

## Stochastic programming modelling languages

The `STOCHPLAM` (Altenstedt, 2001) and `SAMPL` (Fourer, 1996) extensions to existing deterministic modelling languages have been proposed which implement different ways of expressing the extra information necessary to define a stochastic programming problem. Their emphasis has been on using a *fixed* event tree, for which the only extra information required by the problem generator is the knowledge of at what future time stochastic

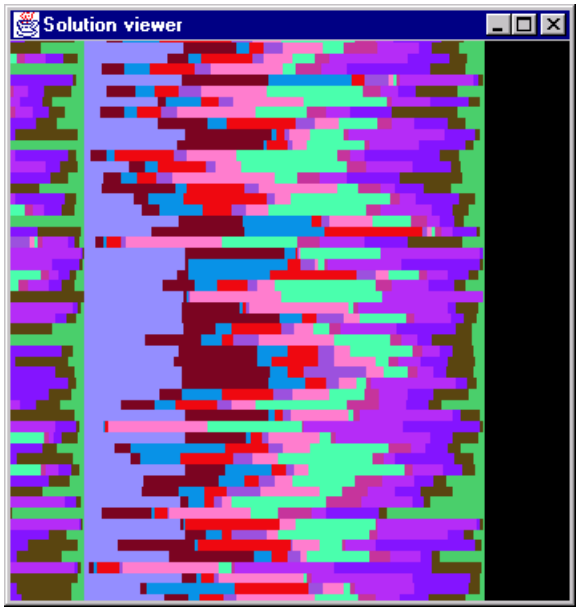


Figure 7: Zoom in on Figure 6.

parameters values become known, variables are chosen and constraints are imposed. In AMPL this can be done either by using the 'suffices' facility to attach extra information to model entities (as described in Section 5) or by imposing additional structure on the syntax of the model. We are currently examining both of these approaches to investigate whether these attempts to extend existing deterministic modelling languages are adequate or whether a purpose-built stochastic programming modelling language is needed as part of the STOCHASTICS™ system under development.

# References

Altenstedt, F. (2001). Stochplam. `http://www.cs.chalmers.se/~alten/stoplam/stochplam.html`.

Archibald, T. W., C. S. Buchanan, K. I. M. McKinnon, and L. C. Thomas (1999). Nested benders decomposition and dynamic programming for reservoir optimization. *Journal of the Operational Research Society 50*, 468–479.

Birge, J. R. (1985). Decomposition and partitioning methods for multi-stage stochastic linear programs. *Operations Research 33*, 989–1007.

Birge, J. R., M. A. H. Dempster, H. I. Gassmann, E. A. Gunn, A. J. King, and S. Wallace (1987). A standard input format for multiperiod stochastic linear programs. In *Committee on Algorithms Newsletter*, Volume 17, pp. 1–20. Mathematical Programming Society.

Chen, Z., G. Consigli, M. A. H. Dempster, and N. Hicks-Pedrón (1998). Towards sequential sampling algorithms for dynamic portfolio management. In C. Zopounides (Ed.), *Operational Tools in the Management of Financial Risks*, pp. 197–211. Dordrecht: Kluwer Academic Publishers.

Consigli, G. and M. A. H. Dempster (1998a). The CALM stochastic programming model for dynamic asset-liability management. In J. M. Mulvey and W. T. Ziemba (Eds.), *World Wide Asset and Liability Modelling*, pp. 464–500. Cambridge University Press.

Consigli, G. and M. A. H. Dempster (1998b). Dynamic stochastic programming for asset-liability management. *Annals of Operations Research 81*, 131–162. Proceedings of the APMOD95 Conference, Brunel University of West London.

Dempster, M. A. H. (1988). On stochastic programming. II. Dynamic problems under risk. *Stochastics 25*(1), 15–42.

Dempster, M. A. H. (1993). CALM: A Stochastic MIP Model. Technical report, Department of Mathematics, University of Essex.

Dempster, M. A. H. (1998). Sequential importance sampling algorithms for dynamic stochastic programming. Technical report, WP 32/98, Judge Institute of Management, University of Cambridge.

Dempster, M. A. H., N. Hicks-Pedrón, E. A. Medova, J. E. Scott, and A. Sembos (2000). Planning logistics operations in the oil industry. *Journal of the Operational Research Society 51*(11), 1271–1288.

Dempster, M. A. H. and R. T. Thompson (1998). Parallelization and aggregation of nested benders decomposition. *Annals of Operations Research 81*, 163–187.

Dempster, M. A. H. and R. T. Thompson (1999). EVPI-based importance sampling solution procedures for multistage stochastic linear programmes on parallel MIMD architectures. *Annals of Operational Research 90*, 161–184.

Escudero, L. F., F. J. Quintana, and J. Salmeròn (1999). CORO, a modelling and algorithmic framework for oil supply, transformation and distribution optimization under uncertainty. *European Journal of Operational Research 114*, 638–656.

Fourer, R. (1996). Proposed new AMPL features: Stochastic programming extensions. `http://www.ampl.com/cm/cs/what/ampl/NEW/FUTURE/stoch.html#scens`.

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

Higle, J. L. and S. Sen (1996). *Stochastic Decomposition*. Dordrecht: Kluwer Academic Publishers.

Infanger, G. (1994). *Planning Under Uncertainty (Solving Large-Scale Stochastic Linear Programs)*. The Scientific Press Series. Boston, MA: Boyd and Fraser.

Infanger, G. (1997). *DECIS User's Guide*. 1590 Escondido Way, Belmont, CA 94002.

King, A. J. (1994). *SP/OSL Version 1.0, Stochastic Programming Interface User's Guide*. Yorktown Heights, NY: IBM T. J. Watson Research Center.

Lane, M. and P. Hutchinson (1980). A model for managing a certificate of deposit portfolio under uncertainty. In *Stochastic Programming*, pp. 473–493. Academic Press.

Marti, K. and P. Kall (Eds.) (1995). *Stochastic Programming: Numerical Techniques and Engineering Applications*. Lecture Notes in Control and Information Sciences. Berlin: Springer-Verlag.

Mulvey, J. and A. Ruszczyński (1995). A new scenario decomposition method for large-scale stochastic optimization. *Operations Research 43*, 477–490.

Rockafellar, R. T. (1976). Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of Operations Research 1*, 97–116.

Rockafellar, R. T. and R. J.-B. Wets (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research 16*, 119–147.

Ruszczyński, A. (1986). A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming 35*, 309–333.

Ruszczyński, A. (1992). Augmented Lagrangian decomposition for sparse convex optimization. Technical Report WP-92-75, International Institute for Applied Systems Analysis, A-2361 Laxenburg, Austria.

Ruszczyński, A. and A. Świętanowski (1995). On the regularized decomposition method for stochastic programming problems. See Marti and Kall (1995), pp. 93–108.

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

Wilkie, A. D. (1987). Stochastic investment models – theory and applications. *Insurance: Mathematics and Economics 6*, 65–83.

Wilkie, A. D. (1995). More on a stochastic asset model for actuarial use. *British Actuarial Journal 1*, 777–964. (Presented to the Institute of Actuaries, 24 April 1995).