# Design and Architecture of a Stochastic Programming Modelling System

*A thesis submitted for the degree of Doctor of Philosophy*

by

Christian Valente

Department of Mathematical Sciences, Brunel University

School of Information Systems, Computing and Mathematics

Brunel University

April 2011

# Abstract

Decision making under uncertainty is an important yet challenging task; a number of alternative paradigms which address this problem have been proposed. Stochastic Programming (SP) and Robust Optimization (RO) are two such modelling approaches, which we consider; these are natural extensions of Mathematical Programming modelling. The process that goes from the conceptualization of an SP model to its solution and the use of the optimization results is complex in respect to its deterministic counterpart. Many factors contribute to this complexity: (i) the representation of the random behaviour of the model parameters, (ii) the interfacing of the decision model with the model of randomness, (iii) the difficulty in solving (very) large model instances, (iv) the requirements for result analysis and performance evaluation through simulation techniques. An overview of the software tools which support stochastic programming modelling is given, and a conceptual structure and the architecture of such tools are presented. This conceptualization is presented as various interacting modules, namely (i) scenario generators, (ii) model generators, (iii) solvers and (iv) performance evaluation. Reflecting this research, we have redesigned and extended an established modelling system to support modelling under uncertainty. The collective system which integrates these otherwise disparate set of model formulations within a common framework is innovative and makes the resulting system a powerful modelling tool. The introduction of scenario generation in the ex-ante decision model and the integration with simulation and evaluation for the purpose of ex-post analysis by the use of workflows is novel and makes a contribution to knowledge.

# Acknowledgements

First of all, I would like to express my gratitude to all my family: my parents Eros and Cristina, for their support, their encouragement and their endless patience, that made them endure my stochastic behaviour. And let's not forget their recent attempts to perturb my concentration, luckily with no consequences. Patrick - my brother - for all the discussions, the fun, the hard DIY, the invitations to poorly cooked dinners and for getting me to live this nice experience here in UK, and of course for providing me with a new family here: Olatz, Ewan and Maren. Grand-mothers (Eliana and Marisa) for their endless, unconditional love and for giving me my beloved roots. Aunts, uncles, cousins, they all were always there. I thank them all, and I do it in Italian with a *"grazie a tutti, di cuore, ora é un po' come aveste un dottorato anche voi: il secondo".*

My supervisors Prof. Gautam Mitra and Dr. Cormac Lucas have been a source of un-countable advice, inspiration and have kept my interest up for all these years. The Mathematics department in Brunel and the company OptiRisk Systems have been thriving environments to work and live in, all the staff always supportive and help-ful, and I thank them for this. My fellow students (now almost all doctors) Luka Jalen, Katharina Schwaiger, Bruno Flach, Christina Erlewein, Gareth Clews, Victor Zviarovich, Carola Kruse (and all the others I did not mention) have been in one way or the other sharing with me this experience, and I definitely could have not done it without them.

Last but not least those who have always been around me, giving me stimulus to cultivate interests in the most diverse fields, sharing surreal experiences, part tak-ing in music performances which were vital for my wellbeing, and usually triggering hilarious laughs: my scattered, but nevertheless close, friends. A (necessarily brief) list would be unfair to the "not fitting" but, as life is often so, I will include it any-way: Filippo, Michele, Manuel, Svava, Marta, Stefano, Giulia, Fabrizio, J. W. Mortazza, Marco, Lukas, Francesca, Sara, Elisa, Matteo, Luka, Ana, Katharina, Christine... and all the others: thanks, you make me what I am, and you bear with the result.

# Contents

**Abstract** ..... **ii**

**Acknowledgements** ..... **iii**

**List of tables** ..... **vi**

**List of figures** ..... **vii**

**Nomenclature** ..... **ix**

**Chapter 1 Introduction and Background** ..... **1**

1.1 Optimum decision making under uncertainty ..... 1

1.2 Stochastic Programming Models ..... 4

1.3 Deterministic Equivalents for SP, CC, ICC, Robust Optimization models ..... 26

1.4 Applications of Stochastic Programming ..... 34

1.5 Scenario Generation ..... 36

1.6 An architecture for an SP modelling system ..... 39

1.7 Outline of the thesis ..... 42

**Chapter 2 Software tools for Stochastic Programming** ..... **44**

2.1 An Information Technology framework for SP ..... 45

2.2 Algebraic Modelling Languages ..... 46

2.3 Extensions of Modelling Languages for SP ..... 49

2.4 Alternative representations ..... 56

2.5 Modelling languages perspective: a tour on AMPL and SAMPL ..... 57

2.6 Conclusions ..... 68

**Chapter 3 Requirements and characteristics of SP solvers** ..... **69**

3.1 Instance level formats ..... 69

3.2 Deterministic equivalent ..... 76

3.3 Decomposition techniques ..... 78

3.4 Model generation ..... 82
3.5 Solver architecture and interface ..... 84
3.6 Conclusions ..... 89
**Chapter 4 Requirements for a Scenario Generation library ..... 91**
4.1 Scenario Generators: a modelling perspective ..... 92
4.2 Overview of SG methods ..... 94
4.3 Desirable properties ..... 100
4.4 An abstract view of SGs ..... 102
4.5 SG Library: an IT perspective ..... 106
4.6 Language constructs for the SG library ..... 116
4.7 Conclusions ..... 118
**Chapter 5 A workflow approach to the investigation of SP models ..... 119**
5.1 Workflows and workflow management systems ..... 120
5.2 Chosen workflow formalisation ..... 123
5.3 Atomic operations in an investigation framework ..... 126
5.4 Example Cases ..... 131
5.5 Conclusions ..... 135
**Chapter 6 Conclusions ..... 136**
6.1 Summary ..... 136
6.2 Contributions of the thesis ..... 137
6.3 Future work ..... 139
**References ..... 140**
**Appendix A Connect a Scenario Generator developed in MATLAB ..... 159**
**Appendix B SAMPL syntax for random parameter declaration ..... 166**
**Appendix C ALM Model ..... 167**

# List of tables

Table 1 Personal planning model entities ..... 6
Table 2 Personal planning model: deterministic productivity ..... 9
Table 3 Personal planning model: single stage productivity realizations ..... 11
Table 4 Personal planning model: two stage productivity realizations ..... 13
Table 5 Personal planning model: mood forecast realizations ..... 16
Table 6 Bundles for example event tree ..... 28
Table 7 Added information for robust optimization problems ..... 33
Table 8 List of SG methods and applications ..... 39
Table 9 Language requirements for scenario based recourse problems ..... 54
Table 10 Language requirements for distribution based SP problem ..... 54
Table 11 Performance of various algorithms ..... 81
Table 12 SG abstraction steps ..... 102
Table 13 Functions used for SG discovery ..... 109
Table 14 Scenario generator example ..... 117
Table 15 ALM model entities ..... 167

# List of figures

Figure 1 Paradigms for modelling under uncertainty ..... 4
Figure 2 Personal planning model: Deviation definition constraint ..... 7
Figure 3 Personal Planning: Single stage data and solution structure ..... 11
Figure 4 Personal planning model: Two stage data and solution structure ..... 14
Figure 5 Personal planning model: multistage data and decision structure ..... 16
Figure 6 Event tree ..... 28
Figure 7 High level overview of a SP modelling system ..... 41
Figure 8 Research areas vs software components ..... 46
Figure 9 Components in an algebraic model ..... 48
Figure 10 Combined Paradigm ..... 53
Figure 11 Extended constructs for SP ..... 56
Figure 12 Personal planning model: Screenshot from SG ..... 63
Figure 13 Instance level formats' role in a modelling system ..... 70
Figure 14 Model generation statistics ..... 83
Figure 15 Comparison of memory usage in model generation ..... 84
Figure 16 Mapping between model classes and solution methods ..... 86
Figure 17 Interface between SPInE and FortSP ..... 87
Figure 18 Link between Scenario Generators and Modelling System ..... 93
Figure 19 Structural compliance between model and SG ..... 94
Figure 20 Scenario generation methods ..... 96
Figure 21 Scenario generator functional perspective ..... 103
Figure 22 Scenario generators as black boxes ..... 103
Figure 23 Abstract representation of a scenario generator ..... 104
Figure 24 Abstract representation of a SG library, single random parameter ..... 105
Figure 25 Abstract representation of a SG library, multiple random parameters ..... 106
Figure 26 IT view of Scenario Generation ..... 107
Figure 27 UML schema of metadata ..... 108
Figure 28 SG discovery screenshot ..... 110
Figure 29 Scenario Generator Interfaces ..... 113
Figure 30 Sequence of interactions MS-SG ..... 115

Figure 31 Petri net model of a task ..... 122
Figure 32 Single activity SWM ..... 123
Figure 33 Concatenation of SWMs ..... 124
Figure 34 SWM with join and split activities ..... 124
Figure 35 SWM with loops ..... 125
Figure 36 Activity 1: Choose decision model ..... 128
Figure 37 Activity 2: Generate Scenarios ..... 128
Figure 38 Activity 3: Generate and solve model ..... 129
Figure 39 Activity 4: Fix variables ..... 129
Figure 40 Activity 5: Collect information ..... 130
Figure 41 Activity 6: Result Analysis ..... 130
Figure 42 Decision evaluation schema ..... 132
Figure 43 SG in sample stability workflow ..... 133
Figure 44 SG out of sample stability workflow ..... 134
Figure 45 Out of sample and backtesting workflow ..... 135
Figure 46 Fund balance constraint ..... 168
Figure 47 Event tree for two-stage formulation ..... 168
Figure 48 Event tree for multi stage ALM model ..... 168

# Nomenclature

| | |
|---|---|
| AML | Algebraic Modelling Language |
| AMPL | A Mathematical Programming Language |
| AR | Auto Regressive |
| ARCH | Auto Regressive Conditional Heteroskedasticity |
| ARMA | Auto Regressive Moving Average |
| CCP | Chance Constraint Programming |
| CSaR | Conditional Surplus at Risk |
| CVaR | Conditional Value at Risk |
| DETEQ | Deterministic Equivalent |
| DSS | Decision Support System |
| EEV | Expectation of the Expected Value |
| EV | Expected Value |
| EVPI | Expected Value of Perfect Information |
| GARCH | Generalized Auto Regressive Conditional Heteroskedasticity |
| GBM | Geometric Brownian Motion |
| HMM | Hidden Markov Model |
| HN | Here and Now |
| ICCP | Integrated Chance Constraint Programming |
| IP | Integer Programming |
| IT | Information Technology |
| LP | Linear Programming |
| MA | Moving Average |
| MIP | Mixed Integer Programming |
| MP | Mathematical Programming |
| MPS | Mathematical Programming System |
| MS | Modelling System |
| NLP | Non Linear Programming |
| OS | Optimization Services |
| OSiL | Optimization Services Instance Protocol |
| OSxL | Optimization Services Protocols |
| QP | Quadratic Programming |

| | |
|---|---|
| RO | Robust Optimisation |
| SAMPL | Stochastic AMPL |
| SG | Scenario Generator |
| SLP | Stochastic Linear Programming |
| SMIP | Stochastic Mixed Integer Programming |
| SNLP | Stochastic Non Linear Programming |
| SOAP | Simple Object Access Protocol |
| SOCP | Second Order Cone Programming |
| SP | Stochastic Programming |
| SSOCP | Stochastic Second Order Cone Programming |
| VSS | Value of Stochastic Solution |
| WF | Workflow |
| WFMS | Workflow Management Systems |
| WS | Wait and See |
| XML | eXtensible Markup Language |

`

# Chapter 1 Introduction and Background

*"God does not play dice" (Albert Einstein, paraphrased from a letter to Max Born, 4 December 1926)*

## 1.1 Optimum decision making under uncertainty

*Optimum decision making* is concerned with the general problem of computing an optimal decision by taking into consideration parameters, their uncertainties and restrictions relevant to it. A significant aspect of moving from a qualitative approach to a quantitative approach is the introduction of Mathematical Programming (MP) paradigm.

Mathematical Programming models enable the modeller to quantify the effects of the decision in terms of the objectives set by the decision maker and these model are formulated to ensure that the decision does not violate any of the restrictions. Such models express the objectives as functions of the *decision variables*, which are restricted to take values on certain domains. The objective functions are maximized or minimized; in case of only one objective function, the problem is called a single objective problem and the solution consists of a set of values of the decision variables which represent the optimum decision, in case of more than one objective function, the problem is referred to as a multi-objective problem, and the product of the optimization is a Pareto efficient set of values of the decision variables, which represent trade-offs between the values of all objective functions.

In this thesis, we restrict the scope to single objective optimization problems, however, most concepts and results are easily extended to a multi-objective context.

A single objective optimization problem is expressed as shown below.

Given a function

$$f: A \rightarrow \mathbb{R} \qquad \textbf{1.1}$$

The computational model is to search (for a minimization problem), for an element $x_0 \in A$ such that $f(x_0) \leq f(x)$ for all $x \in A$.

`

MP models can be further classified by two main criteria: the form of the domain $A$ and the type of the decision variables and the objective function.

Linear Programming (LP) models are characterised by constraints and an objective function which are linear combination of the decision variables while Quadratic Programming (QP) models can have quadratic terms in the objective function. Depending on the type of decision variables, the optimization models are classified as Integer Programming (IP) models if all the variables are integer or Mixed Integer Programming (MIP) models if some of the variables are integers and the remaining are continuous. Following the criteria regarding the form of $A$ we have Second Order Cone Programming (SOCP) models if some constraints are quadratic, see (Lobo et al., 1998) and generic Conic Optimization models if $A$ is a convex cone.

Linear Programming (LP) models were first introduced in the 40's, by early researchers such as Kantorovich, Dantzig and Von Neumann. LP has found many applications in diverse fields involving planning and scheduling. Typical examples are supply chain logistics, financial planning and despatch of energy (electricity or gas). The enormous growth in computational power, as well the development of new mathematical techniques and software tools to formulate and solve these classes of problems has made it possible to apply these models to real world situations in which the number of variables and constraints can be very large.

It is out of the scope of this thesis to provide a comprehensive history of all MP paradigms; what is important is that their success has showed up their limitations. A fundamental assumption for this class of decision models is that the parameters which define the models are known with certainty. This assumption of certain knowledge (deterministic) in many cases does not hold.

Consider for example the future commodity prices or interest rates in a financial planning model, the hourly energy demand in an energy network problem or the demand for a particular characteristic in a blending model: assuming these parameters are known with certainty at solution time could lead to solutions that are not optimal or even not feasible in the real world.

`

In the field of *optimum decision making under uncertainty*, the assumption of a deterministic world is relaxed and different procedures and paradigms arise.

A first step in applying MP techniques to a non-deterministic setup is to consider parameters estimation as central in the modelling process, however, in spite of much care put into forecasting the outcome of non-deterministic events, the forecast could always be incorrect or not precise. The modeller needs therefore to take into account the effects introduced by the uncertainty into the underlying optimisation models and study how the inevitable defects in the forecasting process can affect the quality of the solution obtained.

*Sensitivity analysis* is therefore introduced to study the effect on the solution obtained for changes in the parameter values used. It is presented in greater detail by many researchers, among which (Arriola & Hyman, 2009). This approach has drawbacks and limitations, as discussed in (Higle & Wallace, 2003), (Wallace, 2000) and (Dupačová, 2002).

Many researchers have postulated MP paradigms such as *Stochastic Programming*, *Dynamic Programming* and *Robust Optimisation* to consider parameter variations. In these approaches the modeller can make better use of assumptions he is able to make about the uncertainty related to the problem. In many cases, it is possible to model the uncertainty itself by means of probability distributions. Stochastic Programming models and Dynamic Programming models make use of this added information about the uncertainty to provide optimal decisions which hedge against future uncertainties. Robust Optimisation is an alternative uncertainty aware modelling paradigm in which very few assumptions about the distributions are made, but nevertheless leading to solutions which are stable in respect to the uncertain future outcomes. In this thesis we focus mainly on Stochastic Programming, however, some aspects of Robust Optimisation are also considered and presented, as the implemented software system provides language features to support the formulation of problem classes with uncertain parameters.

`

## 1.2 Stochastic Programming Models

Stochastic Programming (SP) is now a well-established approach to decision making under uncertainty; SP requires the explicit inclusion of the uncertainty in the form of probability distributions of the model parameters which are random. The SP models can be further categorized by the way in which the uncertainty is expressed and dealt with in the underlying optimisation model.

The classification of stochastic programming problems shown in Figure 1 is based on the taxonomy proposed by (Gassmann & Ireland, 1996). It has been extended with the introduction of the Expected Value models, the problems with Chance Constraints and the problems with Integrated Chance Constraints. The alternative paradigms (Dynamic Programming and Robust Optimisation) are shown aside, to contextualise them in the broad area of modelling under uncertainty.

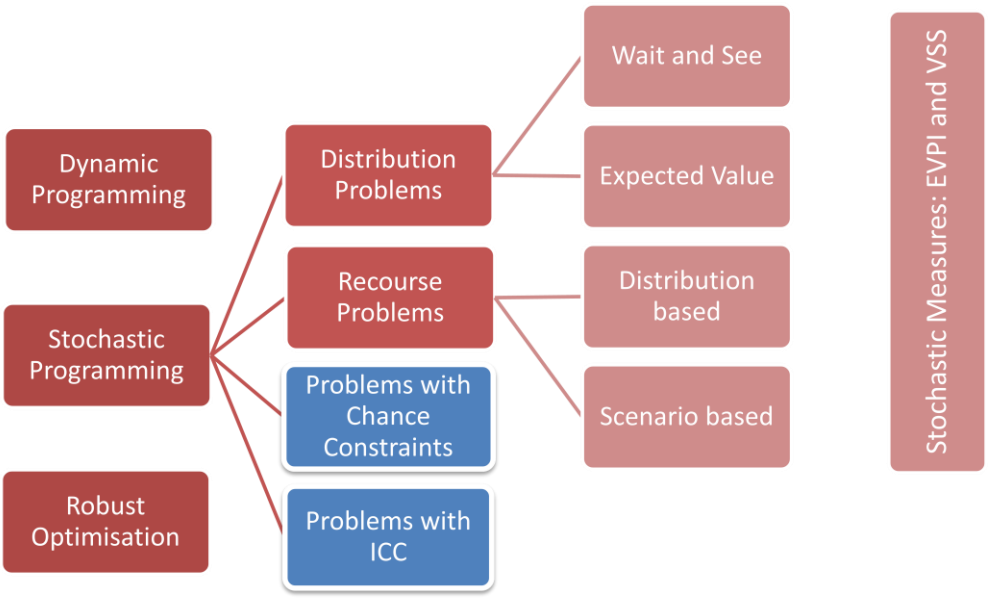



**Figure 1 Paradigms for modelling under uncertainty**

`

### *Definition of a personal planning model*

A simple example has been devised to illustrate the different classes of models in this chapter. For each class of SP problems introduced, a subsection titled *Example* will contain some considerations in reference to this model, formulated in the considered class. The first formulation is deterministic, as a linear optimization problem, and reads as follows.

This model focuses on personal time management. The subject is a person, who has various duties *D* to perform (i.e. a study related project and a work related one) and to finish by the end of the time horizon (the time set is *T*, and its members are the days which have to be planned).

The duties require an amount of work units each ($WR_D$) ; work units are completed allocating work hours to that task ($W_{dt}$).

The subject has different productivity ($P_{dt}$) in terms of work units done per hour in the different tasks (he might be consider better talented for one or the other duty) and also the productivity changes among the planning horizon.

The aim is to minimize the stress, which is modelled as proportional to the amount of overwork $O_t$ the subject undertakes each day (the over-deviation on the average working hours $HA$) and is diminished by under-work $U_t$ (amount of hours devoted to other, relaxing activities).

In each day which is part of the time horizon, the subject has to decide how many hours to work on each task, taking care not to avoid the maximum allowed working hours a day $H$.

Table 1 below shows a possible list of the entities for this model.

`

| Type | Name | Notation | Description | Dimensions | Unit of measure |
|---|---|---|---|---|---|
| Indices (sets) | *DUTIES* | $D$ | Duties to be done | | |
| | *TIME* | $T$ | Time periods | | |
| Parameters (data) | *Wakin Hours* | $H$ | Max work hours per day | | [hours] |
| | *StressFactor* | $S$ | Stress factor of overwork | | [hours$^{-1}$] |
| | *RelaxFactor* | $R$ | Relax factor of underwork | | [hours$^{-1}$] |
| | *Work Required* | $WR_d$ | Work units required | DUTIES | [units] |
| | *Productivity* | $P_{dt}$ | Productivity | DUTIES, TIME | [units/hour] |
| | *Average Work* | $HA$ | Average work hours per day | | [hours] |
| Variables | *Work* | $W_{dt}$ | Amount of work to undertake | DUTIES, TIME | [hours] |
| | *UnderWork* | $U_t$ | Amount of rest | TIME | [hours] |
| | *OverWork* | $O_t$ | Amount of overwork | TIME | [hours] |

**Table 1 Personal planning model entities**

*Objective function*

The objective of the model is to minimize the stress level of the person on all the planning horizon. It is defined as increased, by a factor, by the number of hours of overwork each day and decreased, by another factor, by the number of hours of underwork (relax).

$$min \sum_{t \in T} (O_t S - U_t R) \qquad \textbf{1.2}$$

*Day Length constraints*

Each day in the planning horizon, there is a maximum number of hours the subject can work. It can be assumed to be determined by the physical length of the day or by health regulations. The model would try to allocate as many working hours as

`

possible on the days in which productivity is high; this constraint serves to make the solution respect the length of the days, putting a hard limit to the total work undertaken in each day.

$$\sum_{d \in D} W_{dt} \leq H \,, \forall t \in T \qquad \textbf{1.3}$$

*Deviation Definition constraints*

This constraint is what defines the amount of work below and over the average ($U_t$ and $O$) that the subject undertakes every day. Being the two named variables defined as positive, if there is any variation between the total amount of work allocated for the day and the average working hours, it will be stored in these variables by the constraint, as expressed in equation 1.4.

$$\sum_{d \in D} W_{d,t} = HA + O_t - U_t \,, \forall t \in T \qquad \textbf{1.4}$$

Figure 2 below illustrates the concept of over and under work.

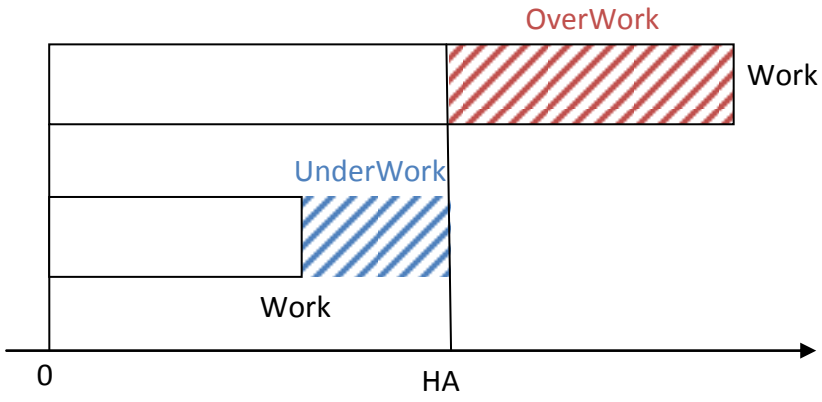



**Figure 2 Personal planning model: Deviation definition constraint**

*Task completion constraint*

Each task has to be completed at the end of the time horizon, by means of allocating working hours to them. The amount of work units completed for each hour depends by the productivity $P_{dt}$, which is here a multiplier of the amount of work hours allocated. The constraint is expressed as in equation 1.5.

$$\sum_{t \in T} W_{dt} P_{dt} = WR_d \,, \forall d \in D \qquad \textbf{1.5}$$

`

## *Linear programming*

These problem classes can be illustrated by first considering the linear programming problem:

$$\begin{gathered} Z = \min cx \\ \text{subject to } Ax = b \\ x \geq 0 \\ \text{where } A \in \mathbb{R}^{m \times n}; c, x \in \mathbb{R}^{n}; b \in \mathbb{R}^{m} \end{gathered} \qquad \textbf{1.6}$$

Let P denote a probability distribution, $\mathcal{F}$ be a sigma field and the triplet $(\Omega, \mathcal{F}, P)$ be a probability space where $\omega \in \Omega$ denote the realizations of the uncertain parameters. Let the realizations of *A*, *b*, *c* for a given event $\omega$ be defined as:

$$\xi(\omega) \; or \; \xi_{\omega} = (A, b, c)_{\omega} \qquad \textbf{1.7}$$

The associated probabilities of these realizations are often denoted as $p(\xi(\omega))$ or $p_{\xi(\omega)}$. For notational convenience these probabilities are denoted simply as $p(\omega)$.

For the same reason, let the feasible regions corresponding to the problem stated in 1.6 and 1.7 be defined as:

$$F^{\omega} = \{x | Ax = b, x \geq 0\} \; for \; \xi(\omega) \qquad \textbf{1.8}$$

*Example. Deterministic to stochastic*

The personal planning model introduced above can be summarized as follows:

$$\begin{gathered} min \sum_{t \in T} (O_t S - U_t R) \\ subject \; to \quad \sum_{d \in D} W_{dt} \leq H \, , \forall t \in T \\ \sum_{d \in D} W_{dt} \leq HA + O_t - U_t \, , \forall t \in T \\ \sum_{t \in T} W_{dt} P_{dt} \geq WR_d \, , \forall d \in D \\ W_{dt} \geq 0 \; \forall d, t \, , \; O_t, U_t \geq 0 \; \forall t \end{gathered} \qquad \textbf{1.9}$$

`

This is a deterministic linear problem; it can be formulated and solved directly, using normal LP techniques. It is formulated in Chapter 2 using AMPL and SAMPL; the focus of this chapter is on the introduction of uncertainty only, therefore suffices to say that, given appropriate values to the parameters, it is possible to construct the matrix *A* and the vectors *b* and *c*, then solve the problem via simplex or interior point methods.

Possible values for the productivity are, supposing that the set *T* contains items corresponding to three time periods, and that the set of duties *D* contains the items "work" and "study":

| TimePeriod | work | study |
|---|---|---|
| **1** | 1 | 0.9 |
| **2** | 0.9 | 1.2 |
| **3** | 1.3 | 1.3 |

**Table 2 Personal planning model: deterministic productivity**

The different productivity values in the various time periods can be interpreted as different amount of distractions the person has to cope with each day. A productivity factor of 1 is "nominal" productivity; a factor lower than 1 means that the person is not as productive as he normally is, maybe due to distractions or lack of concentration. A factor greater than 1 can be interpreted as an indicator of a "good day": a day in which the person is very proficient in the task at hand.

Supposing that the productivity $P_{dt}$ is not known with certainty, as the person might not know how good he will perform the next days in advance, the Stochastic Programming approach requires us to model our productivity forecasts for each period in the time horizon. Depending on how the forecast is generated (the information structure) and how the decision variables are reflecting it, this problem can be formulated as a single stage, two stage or multistage SP problem, as shown in the remainder of this section.

In terms of the just introduced theoretical framework, the parameter $P_{dt}$ now depends on the particular forecast we are in it. Since $P_{dt}$ was used in the constraint matrix $A$, the introduction of its dependency from the forecast $\omega$ causes the feasi-

`

ble region $F^{\omega}$ to depend on the particular succession of realizations we have. To optimize the strategy, despite this uncertainty, is the aim of Stochastic Programming.

### *Distribution based versus Scenario based recourse problems*

The problem defined in 1.6, 1.7 and 1.8 is a mathematical programming model with uncertainty about the values of some of the parameters. If the distribution of $\xi(\omega)$ is continuous, the problem is called a *distribution based recourse problem* (Gassmann & Ireland, 1996)*;* except from some trivial cases, such problems cannot be solved. If the distribution is discrete, the cardinality of the support is limited by the available computing power, therefore in most practical applications the distributions of the stochastic parameters have to be approximated by discrete distributions with a limited number of outcomes (Kaut & Wallace, 2003).

In the discrete statement of the problem given by 1.6, 1.7 and 1.8, the event parameter takes the range of values $\omega = 1, \dots, |\Omega|$; there are associated random vector realisations $\xi(\omega)$ and probabilities $p(\omega)$ such that:

$$\sum_{\omega\in\Omega} p(\omega) = 1 \ \textit{and} \ \ \Xi = \bigcup_{\omega\in\Omega} \xi(\omega) \qquad \textbf{1.10}$$

The discretization $\Xi$ is usually called a *set of scenarios*, and its representation following the dynamic structure of the problem is a *scenario* tree. A stochastic problem whose event outcomes are represented by a scenario tree is called *scenario based recourse problem.* In this thesis only this class of models are considered.

### *Stochastic Programming Problems with Recourse (Here and now)*

*a. Single stage SP Problems*

A simple (single stage) stochastic programming model is formulated as:

$$\begin{gathered} Z_{HN} = \min E[c(\omega)x] \\ \textit{where } x \in F \\ \textit{and } F = \bigcap_{\omega\in\Omega} F^{\omega} \end{gathered} \qquad \textbf{1.11}$$

`

The optimal objective function value $Z_{HN}$ denotes the minimum expected costs of the stochastic optimisation problem. The optimal solution $x^*_{HN} \in F$ hedges against all possible events $\omega \in \Omega$ that may occur in the future.

*Example*

In a single stage model, we give scenarios for the productivity parameter; they correspond to different "general moods", ordered from the least productive to the most. They have been obtained simply multiplying the base productivity given in Table 2 by a factor, which was increasing with the mood. The parameter $P_{dt}$ will then be dependent on the scenario we are in, thus becoming $P_{dts}$.

| | s1 | s2 | s3 |
|---|---|---|---|
| **work(1)** | 0.8 | 1 | 1.3 |
| **work(2)** | 0.72 | 0.9 | 1.17 |
| **work(3)** | 1.04 | 1.3 | 1.69 |
| **study(1)** | 0.72 | 0.9 | 1.17 |
| **study(2)** | 0.96 | 1.2 | 1.56 |
| **study(3)** | 1.04 | 1.3 | 1.69 |

**Table 3 Personal planning model: single stage productivity realizations**

This creates productivity scenarios, which can be visualized as in the left part of Figure 3 below.

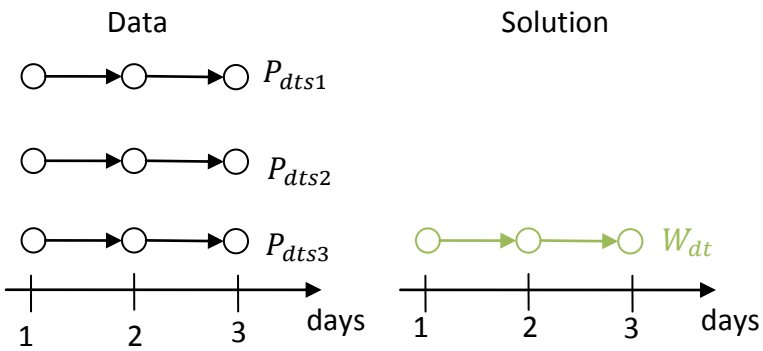



**Figure 3 Personal Planning: Single stage data and solution structure**

An optimal strategy, shown in green in Figure 3, is given by the implementable decision variables $W_{dt}$, and it is a strategy to be followed in each time period and for all

`

scenarios. It optimizes the model 1.12, in which *Sc* is the set of all scenarios and $Pr_s$ is the parameter which stores their probability.

$$\begin{aligned} & min \sum_{t \in T} (O_t S - U_t R) \\ & subject\ to \quad \sum_{d \in D} W_{dt} \leq H\ , \forall t \in T \\ & \sum_{d \in D} W_{dt} \leq HA + O_t - U_t\ , \forall t \in T \\ & \sum_{t \in T} W_{dt} P_{dts} \geq WR_d\ , \forall d \in D, \forall s \in Sc \\ & W_{dt}\ \geq 0\ \forall d, t\ O_t, U_t \geq 0\ \forall t \end{aligned} \qquad \textbf{1.12}$$

The obtained solution $W_{dt}$ is by formulation feasible in each scenario and it does not depend on any dynamically available information on the uncertainty. Furthermore, note that the objective function for this model in this formulation does not consider scenarios at all. This is not generally the case, as the objective function could have parameters which depend on scenarios.

b. Two-stage SP Problems

The classical two-stage SP model with recourse is formulated as:

$$\begin{aligned} Z_{HN} &= \min cx + E_\omega[\, Q(x, \omega)] \\ &\text{subject to } Ax = b \\ & x \geq 0 \end{aligned} \qquad \textbf{1.13}$$

where:

$$\begin{aligned} Q(x, \omega) &= \min f(\omega) y(\omega) \\ &\text{subject to } D(\omega) y(\omega) = d(\omega) + B(\omega) x, \\ & y(\omega) \geq 0, \\ & \omega \in \Omega \end{aligned} \qquad \textbf{1.14}$$

The matrix *A* and the vector *b* are known with certainty. The function $Q(x, \omega)$, referred to as the *recourse function*, is in turn defined by the linear program set out in 1.14. The *recourse matrix* $D(\omega)$, the *right-hand side* $d(\omega)$, the *technology trix* $B(\omega)$, and the objective function coefficients $f(\omega)$ of this model may be random. If the recourse matrix *D* is fixed for all realizations then the problem is known

`

as *SP* problem *with fixed recourse*; if *D* takes the form $D = (I, -I)$, it is known as SP problem *with simple recourse*.

Two-stage Stochastic Programming problems with recourse separate the model's decision variables into first stage and second stage. The dynamic nature of the problem can be easily seen: an optimal first-stage decision *x* is determined such that it is feasible for all realisations $\omega \in \Omega$ and has the minimum cost, while the second-stage decision $y(\omega)$ is taken after the outcome $\omega$ is observed, and compensates and adapts to different realisations.

*Example*

In a two stage model, we allow part of the decision variables (the recourse actions, or second stage variables) to depend on the scenario we are in, and therefore react to it. It therefore requires data which distinguish between a first stage in which we know with certainty our parameters, and a second stage in which we have to make forecasts.

We can reuse the data generated for the single stage model, taking care of the fact that at time period 1 all the productivity values should be certain (i.e. equal for all scenarios). In our case, we decide that the person knows what his mood at day 1 is, and can therefore assume his productivity for that day as certain. The resulting data is displayed below.

| | s1 | s2 | s3 |
|---|---|---|---|
| **work(1)** | **1** | **1** | **1** |
| **work(2)** | 0.72 | 0.9 | 1.17 |
| **work(3)** | 1.04 | 1.3 | 1.69 |
| **study(1)** | **0.9** | **0.9** | **0.9** |
| **study(2)** | 0.96 | 1.2 | 1.56 |
| **study(3)** | 1.04 | 1.3 | 1.69 |

**Table 4 Personal planning model: two stage productivity realizations**

The same kind of decision is taken for the decision variables, we therefore allow for decisions at day 2 and 3 to depend on scenario. The resulting data structure and solution structure is displayed in Figure 4 below.

`

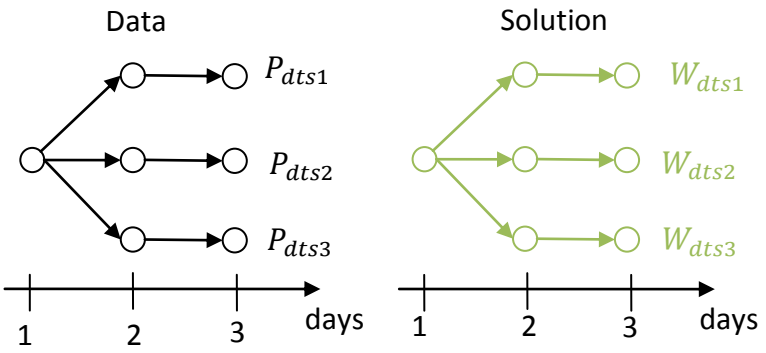



**Figure 4 Personal planning model: Two stage data and solution structure**

The corresponding problem is:

$$
\begin{aligned}
& min \sum_{s \in Sc} Pr_s \sum_{t \in T} (O_{ts}S - U_{ts}R) \\
& subject\ to \quad \sum_{d \in D} W_{dts} \leq H\ , \forall t \in T, \forall s \in Sc \\
& \sum_{d \in D} W_{dts} \leq HA + O_{ts} - U_{ts}\ , \forall t \in T, \forall s \in Sc \\
& \sum_{t \in T} W_{dts} P_{dts} \geq WR_d\ , \forall d \in D, \forall s \in Sc \\
& W_{dts}\ \geq 0\ \forall d, t, s\ O_{ts}, U_{ts} \geq 0\ \forall t, s
\end{aligned}
\qquad \textbf{1.15}
$$

in which it is clearly shown that the decision variables are now scenario dependant. This would not be true for the decision variables at the first time period, as we decided to place them in stage one, but it is often written in this way to simplify the notation. This way of formulating the model is called *explicit non-anticipativity*, as shown in Section 1.3. To enforce the expected behaviour, we use constraints forcing the values of the variables to be equal across all scenarios, for all time periods in which these variables are in stage 1. These constraints are called *non-anticipativity constraints.* For this model, they are written below:

$$
\begin{aligned}
W_{dt1} &= W_{dts}\ \forall d \in D, t = 1, s \in Sc\text{-}\{1\} \\
U_{t1} &= U_{ts}\ \ t = 1, s \in Sc\text{-}\{1\} \\
O_{t1} &= U_{ts}\ \ t = 1, s \in Sc\text{-}\{1\}
\end{aligned}
\qquad \textbf{1.16}
$$

`

c. Multi stage SP Problems

The class of two-stage problems specified by 1.13 and 1.14 can be extended to the *multistage recourse program* considering a more complex dynamic setting: instead of having the two decisions *x* and $y(\omega)$, we consider now T sequential decisions $x_0, x_1, \ldots, x_T$ to be taken at the stages *t = 1, 2, .., T*. The term "stages" can, but need not, be interpreted as "time periods"; although these concepts coincide in many applications, a stage can be regarded in general as a step where new information about the state of nature is provided.

A decision made in stage *t* should be based on the knowledge of the previous decisions and realisations $(x_i, \xi_i \mid i \in 1, .., t-1)$ and such decision only affects the subsequent decisions $(x_i \mid i \in t+1, .., T)$. In Stochastic Programming this concept is known as non-anticipativity and has to be taken into account when formulating the problem in a deterministic equivalent setting. The multistage stochastic programming recourse problem has the form (following (Dempster, 1988), (Ermoliev & Wets, 1988)):

$$
\begin{aligned}
&\min c_1 x_1 + E_{\xi_2}\left[\min_{x_2} c_2 x_2 + \cdots + E_{\xi_T|\xi_{T-1}|..|\xi_2}\left[\min_{x_T} c_T x_T\right]\right] \\
&\textit{subject to}\ A_{11}x_1 = b_1 \\
&\qquad A_{21}x_1 + A_{22}x_2 = b_2 \\
&\qquad A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = b_3 \\
&\qquad \vdots \qquad\qquad \vdots \qquad\qquad \text{1.17} \\
&\qquad A_{T1}x_1 + A_{T2}x_2 + A_{T3}x_3 + \ \ldots\ + A_{TT}x_T = b_T \\
&\textit{with}\quad l_t \le x_t \le u_t\ ,\ t = 1,\ldots,T
\end{aligned}
$$

`

*Example*

As a simple productivity model, we assume that, each day, the person can be either in a good (productive) or a non-productive mood, and the mood influences equally the productivity in all the duties. The mood in each time period is supposed to be independent from all the others.

The proposed forecasts are therefore easily represented in the following way:

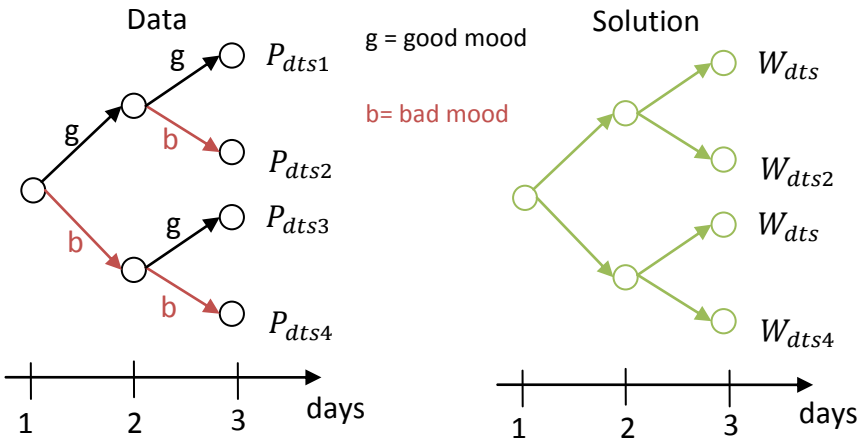



**Figure 5 Personal planning model: Multistage data and decision structure**

Realizations for the productivity parameter can therefore be represented by the following table, where productivity is multiplied by a parameter *g* in case of good mood and by a parameter *b* in case of bad mood.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **work(1)** | 1 | 1 | 1 | 1 |
| **work(2)** | 0.9g | 0.9g | 0.9b | 0.9b |
| **work(3)** | 1.3g | 1.3b | 1.3g | 1.3b |
| **study(1)** | 0.9 | 0.9 | 0.9 | 0.9 |
| **study(2)** | 1.2g | 1.2g | 1.2b | 1.2b |
| **study(3)** | 1.3g | 1.3b | 1.3g | 1.3b |

**Table 5 Personal planning model: mood forecast realizations**

The decision variable at each time period can depend on knowing the information up to that time period; this means that each time period corresponds to a stage. This staging information must be enforced by mean of *non anticipativity constraints*, which are shown below for the variable $W_{dts}$.

`

$$W_{dt1} = W_{dts}\ \forall d \in D, t = 1, s \in Sc\text{-}\{1\}$$
$$W_{dt1} = W_{dt2}\ \forall d \in D, t = 2 \qquad \textbf{1.18}$$
$$W_{dt3} = W_{dt4}\ \forall d \in D, t = 2$$

d. Chance Constraints

The chance-constrained programming problems (CCP) were first introduced in (Charnes & Cooper, 1959). This class of problems deals with the fact that, however in representing a SP problem modellers often use the goal programming approach (i.e. penalties in the objective for violations in the constraints) to account for constraints violations, sometimes it is not possible to quantify the penalty, or penalties cannot be modelled in any reasonable way. The CCP approach considers a decision feasible whenever it is feasible with a high probability. A *probabilistic* or *chance constraint* can be expressed as follows:

$$P(h(x, \xi) \geq 0) \geq p \qquad \textbf{1.19}$$

where *x* and $\xi$ are respectively decisions and random vectors, *P* is a probability measure and $p \in \{0,1\}$ is called the *probability* or *reliability* level.

In a two-stage SP problem with *m* random constraints and defining $I \coloneqq \{1, \dots, m\}$, we distinguish between *individual chance constraints*

$$P(h_i(x, \xi) \geq 0) \geq p_i, \quad i \in I \qquad \textbf{1.20}$$

and *joint chance constraints*

$$P((h_i(x, \xi) \geq 0,\ \ i \in I) \geq p \qquad \textbf{1.21}$$

Chance constraints are inherently a qualitative risk measure, and have been used in a wide range of applications (see, among many others, (Schwaiger, 2009) and references in (van der Vlerk, 1996-2007)) however there are applications in which quantitative risk measures are more appropriate. Another sometimes undesirable characteristic of chance constraints problems is that they are *non-convex* in general; in particular, this is true if the underlying random vector $\omega$ follows a discrete distribution (Dentcheva et al., 2000).

`

*Example*

In the personal production model, we might want to allow for a certain probability of failure in completing the tasks in the predetermined time horizon. Considering the *Task Completion* constraint (see equation 1.5), we can therefore express it as two individual chance constraints, as in:

$$P\left(\sum_{t\in T} W_{dts}P_{dts} \geq WR_d\right) \geq \alpha, \qquad \forall d \in D \tag{1.22}$$

The individual chance constraints 1.22 above guarantee that the subject will reach (independently) the end of each duty, with a probability $\alpha$. The probability of reaching the end of all the tasks is not constrained.

To control the combined probability of failure among all the tasks, the constraint can be reformulated as a *joint* chance constraint, where the decision guarantees that the subject will reach the end of all duties with probability $\alpha$.

$$P\left(\sum_{t\in T} W_{dts}P_{dts} \geq WR_d \ \forall d \in D\right) \geq \alpha, \tag{1.23}$$

e. Integrated Chance Constraints

The arguments given in the paragraph above motivated the research of a different approach; Integrated Chance Constrained Programming has been introduced in (Haneveld, 1986) as an alternative, quantitative and in general convex approach to control and measure feasibility in a SP problem. The ICCP approach considers a problem to be feasible if the *expected violation* of the constraint is less than a predefined value. Integrated Chance Constraints (ICC) are defined in (Haneveld, 1986) as the *individual integrated chance constraints:*

$$E_\omega[\eta_i(x,\omega)^-] \leq \beta_i, \qquad \beta_i \geq 0, \qquad i \in I \tag{1.24}$$

and the *joint integrated chance constraints:*

$$E_\omega[\max_{i\in I} \eta_i(x,\omega)^-] \leq \beta, \qquad \beta \geq 0, \tag{1.25}$$

`

where $\eta_i(x,\omega)^-$ represents the under-deviation that occurs in constraint *i* under realisation $\omega$, and $\beta_i$ is called the *shortfall parameter* and it limits the (maximum) expected shortfall in the (set of) ICCs.

Applications of Integrated Chance Constraints are found in many fields, but their application in finance defines one important class, as it connects to the well-known Conditional Surplus-at-Risk (CSaR, a variant of Conditional Value at Risk (CVaR)), see (Fábián & Veszprémi, 2008).

*Example*

Going back to the same model, we might want to allow for a certain average "amount" of failure in completing the tasks. The amount of failure could be expressed as the amount of  units still needed to complete them. To do so, we can express the *Task Completion* constraint as two individual integrated chance constraints, as in:

$$E\left[\left(\sum_{t\in T}(W_{dts}P_{dts}) - WR_d\right)^-\right] \leq \beta, \qquad \forall d \in D \tag{1.26}$$

The constraints above guarantee that the subject will, in average, not miss more than $\beta$ work units of each task at the end of the planning horizon.

As done for the chance constraints, a Joint Integrated Chance Constraint can be expressed as follows:

$$E\left[\max_{d\in D}\left(\sum_{t\in T}(W_{dts}P_{dts}) - WR_d\right)^-\right] \leq \beta \tag{1.27}$$

This formulation controls the expected maximum units behind schedule among all works to be less than $\beta$.

### *Expected value problem*

The Expected Value (EV) model is constructed by replacing the random parameters by their expected values. Such an EV model is thus a linear program, as the uncertainty is dealt with before it is introduced into the underlying linear optimisation

`

model. It is common practice to formulate and solve the EV problem in order to gain some insight into the decision problem.

Denoting with $X_{EV}^*$ the decision vector resulting from the optimization of the expected value problem, its evaluation against all possible scenarios:

$$Z_{EEV} \stackrel{\text{def}}{=} \sum_{s=1}^{S} p_s * Z_s$$

$$\text{where } Z_s = c_s X_{EV}^*$$

takes the name of Expectation of the Expected Value solution. If there are scenarios s for which $X_{EV}^*$ is not feasible, then $Z_{EEV}$ is set to be ∞.

### *Wait and see problems*

Wait and See (WS) problems assume that the decision-maker is somehow able to wait until the uncertainty is resolved before implementing the optimal decisions. This approach therefore relies upon perfect information about the future; operatively, we solve one separate LP problem for each available scenario, thus obtaining the optimal strategy in each scenario. Because of its very assumptions such a solution cannot be implemented and is known as the "passive approach". Wait and see models are often used to analyse the probability distribution of the objective value. We assign to the expectation of the objective values of all the solved Wait and See models the notation $Z_{WS}$.

$$Z_{WS} \stackrel{\text{def}}{=} \sum_{s=1}^{S} p_s * Z_s$$

$$Z_s = \min c(\omega)x$$

$$where\ x \in F^{\omega}$$

### *Stochastic Measures*

It can be shown that the three objective function values $Z_{WS}, Z_{HN}, Z_{EEV}$ are connected by the following ordered relationship:

$$Z_{WS} \leq Z_{HN} \leq Z_{EEV}$$

`

The inequality:

$$Z_{HN} \leq Z_{EEV}$$

can be argued in the following way: any feasible solution of the average value approximation is already considered in the Here and Now model, therefore the optimal Here and Now objective must be better.

### *The value of the stochastic solution (VSS)*

The difference between these two solutions defines the **Value of the Stochastic Solution** (VSS) for a minimisation problem:

$$VSS \stackrel{\text{def}}{=} Z_{EEV} - Z_{HN}$$

This is a measure of how much can be saved by implementing the (computationally expensive) Here and Now solution as opposed to the deterministic expected value solution. The practical computation of VSS is strictly related to the approach used in the computation of $Z_{EEV}$.

### *The expected value of perfect information (EVPI)*

Another important index is represented by the **Expected Value of Perfect Information** (EVPI):

$$\text{EVPI} \stackrel{\text{def}}{=} Z_{HN} - Z_{WS}$$

This measure of a stochastic optimisation problem is interpreted as the expected value of the amount the decision maker is willing to pay to have perfect information (i.e. knowledge) about the future scenarios. A relatively small EVPI indicates that better forecasts will not lead to much improvement; a relatively large EVPI means that incomplete information about the future may prove costly.

### *Bounds on EVPI and VSS*

Some useful bounds on the EVPI and VSS are presented below:

$$0 \leq EVPI \leq Z_{HN} - Z_{EV} \leq Z_{EEV} - Z_{EV}$$

`

$$0 \leq VSS \leq Z_{EEV} - Z_{EV}$$

These can help in estimating the relative benefit of implementing the computationally costly Stochastic Programming solution, as opposed to approximate solutions obtained by processing the Expected Value LP problem.

### *Robust Optimization*

Modelling a problem following the Stochastic Programming approach requires the analyst to make strong assumptions on the nature of the uncertainty, that is, to supply or postulate probability distributions of the random parameters. There are cases in which it is impossible, or not practical, to give reasonable estimates of these probability distributions but in which the robustness of the solution obtained is vital anyway. The first set of studies which addressed these questions was due to Soyster (Soyster, 1973) and led to a framework which is now established as Robust Optimization (RO).

There are now three well known formulations of RO problems; these are given by the above mentioned Soyster, Ben-Tal and Nemirovski (see (Ben-Tal & Nemirovski, 1998), (Ben-Tal & Nemirovski, 1999), (Ben-Tal & Nemirovski, 2000)) and Bertsimas and Sim (Bertsimas & Sim, 2004). They all share the advantage that minimal assumptions about the nature of the uncertainties have to be made and they differ in respect of the ways in with which they represent the uncertainty sets. More specifically, the formulations by Soyster and by Bertsimas and Sim use polyhedral uncertainty sets, while the formulation by Ben-Tal and Nemirovski considers an ellipsoidal uncertainty set, transforming the original LP problem into a Second Order Cone Programming (SOCP) problem. The solution of these RO problems addresses an important question as to how much optimality for the nominal problem is given up in order to ensure robustness and changes the class of the resulting problem.

`

Consider the following nominal linear optimisation problem:

$$\begin{gathered} Z = \max \underline{c}^T \underline{x} \\ subject\ to \sum_{j=1}^{n} \underline{a}_j \underline{x}_j \leq \underline{b} \\ \underline{l} \leq \underline{x} \leq \underline{u} \\ \text{where } \underline{a}_j, \underline{b} \in \mathbb{R}^m; \underline{c}, \underline{x}, \underline{l}, \underline{u} \in \mathbb{R}^n \end{gathered} \qquad \textbf{1.28}$$

and assume that data uncertainty only affects elements in matrix *A*.

The uncertainty model U we consider is the following:

For a particular row *i* of the matrix *A* let $J_i$ represent the set of coefficients in row *i* that are subject to uncertainty. Each entry $a_{ij}, j \in J_i$ is modelled as a symmetric and bounded random variable $\tilde{a}_{ij}, j \in J_i$ that takes values in $[a_{ij} - \check{a}_{ij}, a_{ij} + \check{a}_{ij}]$, where $\check{a}_{ij} > 0$ is the deviation of variable $\tilde{a}_{ij}$ around its mean value $a_{ij}$. Associated with the uncertain data $\tilde{a}_{ij}$, we define the random variable $\eta_{ij} = (\tilde{a}_{ij} - a_{ij})/\check{a}_{ij}$, which obeys an *unknown but symmetric* distribution, and takes values in [-1,1].

Example Consider two parameters

a. Soyster's Formulation

In general, Soyster's formulation considers the linear optimization problem:

$$\begin{gathered} Z = \max \underline{c}^T \underline{x} \\ subject\ to \sum_{j=1}^{n} \underline{a}_j \underline{x}_j \leq \underline{b} \\ where\ \underline{a}_j \in \begin{Bmatrix} [a_{1j} - \check{a}_{1j}, a_{1j} + \check{a}_{1j}] \\ \dots \\ [a_{mj} - \check{a}_{mj}, a_{mj} + \check{a}_{mj}] \end{Bmatrix} \ j = 1, \dots, n \\ \underline{x} \geq 0. \end{gathered} \qquad \textbf{1.29}$$

This is an adaption from the formulation given in (Bertsimas & Sim, 2004), which was syntactically incorrect.

Soyster shows that the problem in 1.29 is equivalent to

`

$$
\begin{aligned}
Z &= \max \underline{c}^T \underline{x} \\
subject\ to \sum_{j=1}^{n} \underline{\bar{a}}_j \underline{x}_j &\leq \underline{b} \\
\underline{x} &\geq 0 \\
\text{where } \bar{a}_{ij} &= a_{ij} + \check{a}_{ij} \quad \forall\, i,j
\end{aligned}
\tag{1.30}
$$

If the uncertainty sets follow the model U, the robust formulation of 1.28 following 1.30 is as follows:

$$
\begin{aligned}
&\max \underline{c}^T \underline{x} \\
subject\ to \sum_{j} a_{ij} x_j + \sum_{j \in J_i} \check{a}_{ij} y_j &\leq b_i \quad \forall\, i \\
\text{-}y_j \leq x_j &\leq y_j \quad \forall\, j \\
\underline{l} \leq \underline{x} &\leq \underline{u} \\
\underline{y} &\geq 0.
\end{aligned}
\tag{1.31}
$$

It can be shown (Bertsimas & Sim, 2004) that the solution to the problem above remains feasible for all realizations $\tilde{a}_{ij}, j \in J_i$, although concerns have been raised regarding the fact that it trades too much of the optimality of the nominal problem to gain this robustness.

b. Ben-Tal and Nemirovski

Considering the problem set out in 1.28, the following robust problem is constructed (Ben-Tal & Nemirovski, 2000):

$$
\begin{aligned}
&\max \underline{c}^T \underline{x} \\
subject\ to \quad \sum_{j} a_{ij} x_j + \sum_{j \in J_i} \check{a}_{ij} y_{ij} + \sqrt{\sum_{j \in J_i} {\check{a}_{ij}}^2 {z_{ij}}^2} &\leq b_i \quad \forall\, i \\
\text{-}y_{ij} \leq x_j - z_{ij} &\leq y_{ij} \quad \forall\, i, j \in J_i \\
\underline{l} \leq \underline{x} &\leq \underline{u} \\
\underline{y} &\geq 0.
\end{aligned}
\tag{1.32}
$$

If the uncertainty is represented by the model U, the probability that the *i* constraint is violated is at most $e^{-\Omega_i^2/2}$; furthermore the model is proven to be less conservative than 1.31 as every solution of the latter problem is feasible to the former problem.

`

c. Bertsimas and Sim

In this approach, a parameter $\Gamma_i$ is introduced, that intuitively controls the trade-off between robustness and optimality of the solution. The problem, in its equivalent linear formulation, is set out below (Bertsimas & Sim, 2004):

$$\begin{gathered} \max \underline{c}^T \underline{x} \\ subject\ to \quad \sum_j a_{ij}x_j + z_i\Gamma_i + \sum_{j\in J_i} p_{ij} \le b_i \quad \forall\, i \\ z_i + p_{ij} \ge \check{a}_{ij}y_j \ \ \forall\, i, j \in J_i \\ -y_j \le x_j \le y_j \ \ \forall\, j \\ l_j \le x_j \le u_j \ \ \forall\, j \\ p_{ij} \ge 0 \ \ \forall\, i, j \in J_i \\ y_j \ge 0 \ \ \forall\, j, z_i \ge 0 \ \ \forall\, i \end{gathered} \qquad \textbf{1.33}$$

Parameter $\Gamma_i$ takes values in the interval $[0, |J_i|]$ and it has the effect of protecting the feasibility of the solution against all cases in which up to $\lfloor\Gamma_i\rfloor$ coefficients $a_{ij}, j \in J_i$ will change, and one coefficient $a_{it}$ changes by $(\Gamma_i - \lfloor\Gamma_i\rfloor)\check{a}_{it}$. More formally, the solution will remain feasible *deterministically* if the realisations behave as specified above, and moreover, even if more than $\lfloor\Gamma_i\rfloor$ parameters change, then the robust solution will be feasible with *very high probability.*

*Example (Soyster formulation)*

Going back to the personal planning example, the planner might not be able to know anything about his future productivity rates but the fact that they will range between around their mean value (which is given by Table 2). We denote the amplitudes of the ranges $\widetilde{P_d}$, which for simplicity is supposed constant in time. This means that for each *t* and *d,* all the productivities are considered as uniformly distributed in $\left[P_{dt} - \widetilde{P_d}, P_{dt} + \widetilde{P_d}\right]$.

A Soyster's formulation of this problem would guarantee that, even in the worst case, all the jobs would be finished on time. That is obtained adding some new variables $y_{dt}$, one for each uncertain parameter. The project completion constraints, one for each task, will then be reformulated as follows:

`

$$\sum_{t\in T} W_{dt}P_{dt} - \sum_{t\in T} y_{dt}\widetilde{P_d} = WR_d \;\forall\, d$$
$$W_{dt} \le y_{dt} \;\; \forall\, d, \forall\, t \qquad \textbf{1.34}$$
$$W_{dt}, y_{dt} \;\ge 0, \forall\, d, \forall\, t$$

A solution to the problem obtained combining 1.9 with the reformulation above will be feasible for each possible future productivities, if they lie in the defined intervals. The logic behind this constraint is simple enough: the amount of work to the aim of satisfying the constraint is calculated with the mean value as $\sum_{t\in T} W_{dt}P_{dt}$. This is then diminished by an amount which correspond to the worse case productivity, calculated using the range $\widetilde{P_d}$ and an additional variable $y_{dt}$. To ensure that the whole constraint represents the worst case productivity, this additional variable is defined to be greater or equal than $W_{dt}$.

## 1.3 Deterministic Equivalents for SP, CC, ICC, Robust Optimization models

Stochastic programming problems are in general harder to model and solve than their deterministic counterparts. In many cases, however, SP models can be transformed into deterministic models, which are investigated using the existing tools for LP, MIP or SOCP, depending on the kind of model. Although this approach suffers a number of drawbacks (see section 2.3), it is worth discussing it here, because it can still be considered the "standard" approach to modelling SP recourse problems, given the current capabilities of the algebraic modelling systems and solution tools. Chance Constrained and Integrated Change Constrained problems do have their deterministic equivalent formulations too, and so do Robust Optimization problems; these models fall outside the classical SP framework, but they support decision making under uncertainty and are therefore considered.

### 1) Stochastic Programming Problems with Recourse

The deterministic equivalent formulation of a stochastic linear program with recourse can be constructed when the distributions of the random parameters are discrete and are provided either explicitly or in form of *scenario data paths*, which define a *scenario tree.* It can be shown that a scenario tree can always be derived

`

from the distributions of individual random parameters, by computing the joint distribution of all random parameters for each stage. Similarly, the distributions of random parameters can be determined given a scenario tree. Thus, a distribution-based recourse problem with discrete and finite probability distributions is transformed into an equivalent scenario-based recourse problem and vice versa.

There are two representations of a stochastic programming problem in deterministic equivalent form, namely the *explicit* (or split-variable) and *implicit* (or compact) representations. The general programming problem with recourse set out in Equation 1.17 is reported below for ease of reading:

$$\begin{aligned} &\min c_1 x_1 + E_{\xi_2}[\min_{x_2} c_2 x_2 + E_{\xi_3|\xi_2}[\min_{x_3} c_3 x_3 + \cdots \\ &\qquad + E_{\xi_T|\xi_{T-1}|\ldots|\xi_2} \min_{x_T} c_T x_T]] \\ &\textit{subject to } A_{11}x_1 = b_1 \\ &A_{21}x_1 + A_{22}x_2 = b_2 \\ &A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = b_3 \\ &\quad . \quad . \\ &\quad . \quad . \quad . \\ &\quad . \quad . \quad . \quad . \\ &A_{T1}x_1 + A_{T2}x_2 + A_{T3}x_3 + \ \ldots \ + A_{TT}x_T = b_T \\ &\qquad \textit{with} \quad l_t \le x_t \le u_t \,,\ t = 1,\ldots,T \\ &\textit{and } \xi_t = (b_t, c_t, A_{t1}, \ldots, A_{tT}) \ \ \forall\, t \in [2,..,T] \end{aligned} \tag{1.35}$$

The uncertainty associated with the random vector $\xi_t$ is represented by a multilevel event tree, where each level is associated with a stage. Figure 6 shows an example of event tree. A scenario s is a path from the root of the event tree to any of the leaves (see scenario 5, highlighted in red in the figure).

`

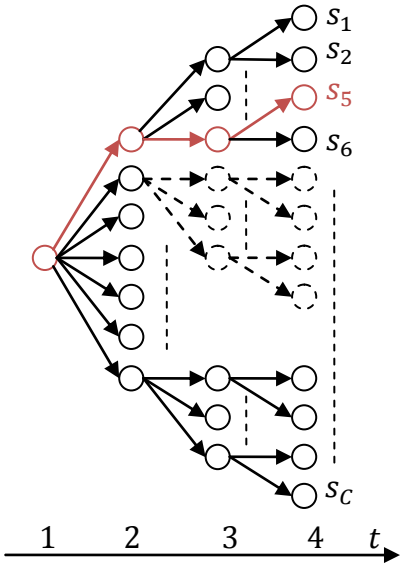



**Figure 6 Event tree**

Let $K_t$ denote the number of nodes in stage $t$, and let $n_{tk}, k \in [1, \dots, K_t], t \in [1, \dots, T]$ denote the *k*-th node of the *t*-th stage of the even tree. A probability $\pi_{tk}$ can be associated with each node $n_{tk}$ such that $\pi_{tk} = P\{\xi_t | \xi_{t-1} | \dots | \xi_2\}$. Hence, arcs in the tree represent the probability distribution of $\xi_t$. The set of scenarios passing through node $n_{tk}$ are identified by $bundle_{tk}$. In the example shown in Figure 6, the number of nodes at each stage is:

$$K_1 = 1,\ K_2 = 7,\ K_3 = 21,\ K_4 = 42.$$

The composition of the bundles for each stage is given in Table 6 below:

| Stage | Bundles |
|---|---|
| 1 | $bundle_{11} = \{1,2,\dots,42\}$ |
| 2 | $bundle_{21} = \{1,2,\dots,6\}, bundle_{22} = \{7,8,\dots,12\}, \dots, bundle_{27} = \{37,38,\dots,42\}$ |
| 3 | $bundle_{31} = \{1,2\}, bundle_{32} = \{3,4\}, \dots, bundle_{3\,21} = \{41,42\}$ |
| 4 | $bundle_{41} = \{1\}, bundle_{42} = \{2\}, \dots, bundle_{4\,42} = \{42\}$ |

**Table 6 Bundles for example event tree**

Given the regular structure of the tree, the bundles of this example are defined in a compact form as:

$$bundle_{tk} = \left\{1 + \frac{S_C}{K_t}(k-1), \dots, \frac{S_C}{K_t}k\right\} \forall t \in 1..T, k \in 1..K_t \qquad \textbf{1.36}$$

`

where $S_C$ is the total number of scenarios. Relation 1.36 is valid for any scenario tree with a constant number of branches at each node of a given stage. In order to build the deterministic equivalent formulation, one needs to consider that the decisions at each stage have to be the same for all scenarios which are indistinguishable up to that stage (that is, the decision are based on the same information, due to the *non-anticipativity*, see Section 1.2). The decision variables of the deterministic equivalent model need to be replicated to reflect the dependency of recourse actions on the event tree, that is, the structure of the problem should allow multiple decisions to be taken when they are based on different information, and vice versa. The choice of the variables replication scheme determines the type of deterministic equivalent model, and determines how the non-anticipativity is enforced in the model.

### *a) Explicit non-anticipativity (split-variable representation)*

In the split variable representation, decisions are replicated for each scenario in each time period. Denoting with $x_{ts}$ the decision at stage $t$ under scenario $s$, the deterministic equivalent problem of equation 1.35 is formulated as follows:

$$\min \sum_{s\in S} \pi_s \sum_{t=1}^{T} c_{ts} x_{ts}$$

$$\begin{aligned} \textit{subject to}\ & A_{11}x_{1s} && = b_1 \\ & A_{21s}x_{1s} + A_{22s}x_{2s} && = b_{2s} \\ & A_{31s}x_{1s} + A_{32s}x_{2s} + A_{33s}x_{3s} && = b_{3s} \\ & \quad . \qquad\qquad . \quad . \quad . && \qquad \forall\, s \in S \\ & A_{T1s}x_{1s} + A_{T2s}x_{2s} + A_{T3s}x_{3s} + \ \ldots\ + A_{TTs} && = b_{Ts} \\ & \textit{with} \qquad l_{ts} \le x_{ts} \le u_{ts}\ ,\ t = 1,\ldots,T \end{aligned} \qquad \textbf{1.37}$$

It is easily seen that the problem defined in 1.37 does not fully capture the structure of an event tree, as all variables are replicated for all scenarios independently by the dependencies between them. Due to the concept of non-anticipativity, some restrictions needs therefore to be added to ensure that if two scenarios $s_i$ and $s_j, i \neq j$, are indistinguishable up to a given time period t, that is, $s_i$ and $s_j$ follow the same path up to stage *t,* then the related decisions (solutions), up to that stage, must also be the same. These restrictions, known as the *non-anticipativity con-*

`

*straints*, are explicitly added to the model, hence the nomenclature *explicit* non-anticipativity. Formally:

$$x_{ts_q} = x_{ts_r}, \qquad \forall\, s_q, s_r \in\ bundle_{tk}, r \neq q, \qquad \forall t \tag{1.38}$$

The non anticipativity constraints for the event tree specified in Figure 6 and Table 6 are therefore:

$$x_{1,1} = x_{1,2} = \cdots = x_{1,64}$$

$$x_{2,1} = x_{2,2} = \cdots = x_{2,7}, \qquad x_{2,8} = x_{2,9} = \cdots = x_{2,13}\ , \ldots, x_{2,37} = x_{2,38} = \cdots = x_{2,42}$$

and similarly for stage 3.

### *b) Implicit non-anticipativity (compact representation)*

The structure given by the scenario tree can be enforced in a different way, defining a reduced set of decision variables for which the non-anticipativity constraints are implicitly satisfied. The compact mathematical formulation of the deterministic equivalent of problem 1.37 is obtained refining the definition of the decision variables, that is, defining $x_{t,n}, n \in \{1, .. K_t\}, \forall t$, as the decision to be taken at time $t$ under all scenarios $s \in\ bundle_{t,n}$. It becomes handy to define the concept of descendants as nodes in the scenario tree which descend from each node $n_{tk}$. We will refer to these sets as $desc_n$.

The mathematical formulation is the following:

$$\min \sum_{s \in S} \pi_s \left( \sum_{t=1}^{T} c_{ts} x_{ts} \right)$$

$$\begin{aligned}
&\textit{subject to}\ \ A_{11}x_{11} && = b_{1,1} \\
&A_{21,n}x_{11} + A_{22,n}x_{2,n} && = b_{2,n} \\
&\qquad \forall\, n \in\ desc_{n11} && \\
&A_{31,n2}x_{11} + A_{32,n2}x_{2,n} + A_{33,n2}x_{3,n2} && = b_{3,n2} \\
&\qquad \forall\, n \in desc_{n_{11}},\ n_2 \in desc_n && \\
&\ .\qquad .\qquad .\qquad .\qquad .\qquad . && \\
&A_{T1,n_T}x_{11} + A_{T2,n_T}x_{2,n} + A_{T3,n_T}x_{3,n2} + \cdots + A_{TT,n_T}x_{T,n_{T-1}} && = b_{3,n_{T-1}} \\
&\qquad \forall\, n \in desc_{n_{11}},\ n_2 \in desc_{n,\ \ldots}, n_{T-1} \in desc_{n_{T-2}} &&
\end{aligned} \tag{1.39}$$

`

$$\textit{with} \qquad l_{t,n} \le x_{t,n} \le u_{t,n} \quad \forall\, t \in [1,\dots,T],\ n \in [1,\dots,K_t]$$

Although formulation 1.39 has a structure which is more compact than formulation 1.37, the explicit definition of non-anticipativity conditions can be exploited for designing specialised algorithms for the efficient solution of the SP problem.

Independently from the chosen formulation, the structure of the event tree must be captured by the model; this fact adds a class of information that is qualitatively different from the usual entities defined in Figure 9 (namely sets, parameters, variables, constraints and objectives) which are used to define a deterministic problem; even though the event tree structure can be expressed in terms of these entities through deterministic equivalent formulations, as we have just explored, it makes sense to preserve the different *quality* of this information, which gives a strong intellectual rationale for the extension of AMLs to stochastic programming. For further reading to the formulation of deterministic equivalent problems, see (Wets, 1974), (Dempster, 1988), (Messina & Mitra, 1997).

### 2) CC problems deterministic equivalent representation

Considering the two-stage SP problem expressed in equations 1.13 and 1.14, its deterministic equivalent formulation can be expressed as:

$$\begin{aligned} & \min c_1 x_1 + \sum_{s=1}^{S} p_s\,(c_{2s} x_{2s}) \\ \textit{subject to} \quad & g_1 \le A_{11}x_1 \le h_1 \\ & g_{2,s} \le A_{21,s}x_1 + A_{22,s}x_{2s} \le h_{2,s} \quad \forall s \in [1,\dots,S] \\ & l_1 \le x_1 \le u_1 \\ & l_{2,s} \le x_{2,s} \le u_{2,s} \quad \forall s \in [1,\dots,S] \end{aligned} \qquad \textbf{1.40}$$

where $p_s$ is the probability of scenario *s* and where the random parameters are supposed to have finite and discrete distributions.

Under these assumptions, an individual chance constraint expressed by equation 1.20 and reported in equation 1.41 below for ease of reading:

$$P(h_i(x,\xi) \ge 0) \ge p_i, \quad i \in I \qquad \textbf{1.41}$$

`

can be formulated, assuming the constraint corresponds to the *i*-th row of matrix $A_{2t,s}$, as:

$$P\{g_2^i \leq A_{21}^i x_1 + A_{22}^i x_2 \leq h_2^i\} \geq \alpha \qquad \textbf{1.42}$$

where $0 < \alpha < 1$ is a reliability level, $g_2^i$ and $h_2^i$ denote the i-th elements of vectors $g_2$ and $h_2$; $A_{21}^i$ and $A_{22}^i$ denote i-th rows of matrices $A_{21}$ and $A_{22}$. It can be shown that the following system of equations is equivalent to 1.42:

$$\begin{aligned}
& g_{2,s}^i \leq A_{21}^i x_1 + A_{22}^i x_2 + M v_s && \forall s \in [1, \dots, S] \\
& A_{21}^i x_1 + A_{22}^i x_2 - M w_s \leq h_{2,s}^i && \forall s \in [1, \dots, S] \\
& v_s \leq z_s && \forall s \in [1, \dots, S] \\
& w_s \leq z_s && \forall s \in [1, \dots, S] \\
& \textstyle\sum_{s=1}^{S} p_s z_s \leq 1 - \alpha && \\
& v_s, w_s, z_s \in \{0,1\} && \forall s \in [1, \dots, S]
\end{aligned} \qquad \textbf{1.43}$$

where M is a suitably chosen large constant, $v_s$, $w_s$ and $z_s$ are additional binary variables. An individual chance constraint with lower and upper limits can therefore be represented at the cost of $2|S|+1$ additional constraints and $3|S|$ additional binary variables (Haneveld & van der Vlerk, 2006).

The deterministic equivalent formulation of chance constraint suffers therefore of the same drawbacks seen in the DEQ formulations of MSSP, namely unnecessary replication of entities and an added complexity in the logic of the model, with the additional penalty of having to deal with binary variables, which can lead to instances that are numerically very difficult to solve.

### 3) ICC problems deterministic equivalent representation

Considering the deterministic equivalent of a two-stage problem as expressed in equation 1.40, and the individual ICC expressed in 1.24 and reported in Equation 1.44 below:

$$E_\omega[\eta_i(x,\omega)^-] \leq \beta_i, \qquad \beta_i \geq 0, \qquad i \in I \qquad \textbf{1.44}$$

it can be formulated, considering $g_2^i = \infty$ for all the realizations of the random parameters, as:

`

$$E\left[\left(h_2^i - A_{21}^i x_1 - A_{22}^i x_2\right)^-\right] \leq \beta \qquad \textbf{1.45}$$

where $\beta \geq 0$ and $(a)^- := max\{-a, 0\}$ is the negative part of $a \in \mathbb{R}$ and represents the negative violation of the *i*-th constraint.

The ICC expressed in 1.45 has the following DEQ form:

$$\begin{gathered} A_{21}^i x_1 + A_{22}^i x_2 - w_s \leq h_{2,s}^i \;\; \forall s \in [1, \dots, S] \\ \sum_{s=1}^{S} p_s \, w_s \leq \beta \end{gathered} \qquad \textbf{1.46}$$

where $w_s$ are additional *continuous* variables, see (Haneveld & van der Vlerk, 2006) and (Ellison et al., 2009) for additional details.

### 4) Robust deterministic equivalent representation

The Robust Optimization problems, in the formulations of Soyster (1.31), Ben-Tal and Nemirovski (Ben-Tal & Nemirovski, 2000) and Bertsimas and Sim (Bertsimas & Sim, 2004) , are already in a format which can be represented respectively by LP, SOCP and LP models. The robust optimization model includes some additional information in respect to the deterministic counterpart. This information, which depends on the specific robust formulation of choice, is listed in Table 7 below:

| Formulation | Added Parameters | Meaning |
|---|---|---|
| **Soyster** | $\boldsymbol{J_i}$ | Uncertainty model U (**scope**) |
| | $\boldsymbol{\breve{a}_{it}}$ | Uncertainty model U (**intervals**) |
| **Ben-Tal and Nemirovski** | $\boldsymbol{J_i}$ | Uncertainty model U (**scope**) |
| | $\boldsymbol{\breve{a}_{it}}$ | Uncertainty model U (**intervals**) |
| | $\boldsymbol{\Omega_i}$ | **Robustness**: the probability that the *i* constraint is violated is at most $e^{-\Omega_i^2/2}$; |
| **Bertsimas and Sim** | $\boldsymbol{J_i}$ | Uncertainty model U (**scope**) |
| | $\boldsymbol{\breve{a}_{it}}$ | Uncertainty model U (**intervals**) |
| | $\boldsymbol{\Gamma_i}$ | **Robustness**: the solution remains feasible if up to $\lfloor \Gamma_i \rfloor$ coefficients $a_{ij}, j \in J_i$ will change, and one coefficient $a_{it}$ changes by $(\Gamma_i - \lfloor \Gamma_i \rfloor)\breve{a}_{it}$. |

**Table 7 Added information for robust optimization problems**

A modeller that would implement a robust formulation without using a specifically designed tool would have to manually implement the somewhat complicated struc-

`

ture of the robust framework of his choice (that is, the added constraints and variables expressed by the systems 1.31, 1.32 or 1.33), on the top of the already hard work which is the construction of the model itself. As the structure of these systems of equations does not change once a formulation is chosen, it makes little or no sense - besides eventually for learning purposes - that the problem owner should personally and manually execute a process that can be fully automated. Hence the language extensions to AMPL we are proposing to express robust optimisation problems (see Section 2.4).

## 1.4 Applications of Stochastic Programming

The adoption of Stochastic Programming as a viable computational modelling paradigm to obtain hedged decisions when the problem is subject to uncertainty has faced many barriers to its adaption. These barriers may be listed as (i) computational complexity and (ii) lack of proper modelling tools to support it. Thirdly (iii) the scepticism of the decision makers in front of the increased complexity of the modelling effort compared to deterministic models combined with estimates of parameters which are uncertain. This is slowly changing and, as the available computational resources increase by the Moore's law, and as algorithms and software to efficiently solve SP problems are made available, practitioners are making use of growing amount of data and information, made available by the evolving IT infrastructure (consider for example the adoption of OLAP cubes instead of relational databases and the ubiquity of data mining techniques). This growing availability of information calls for efficient computational tools despite the increase in available computing power.

Areas in which the application of SP is already common are:

**Finance:** Pioneering works in Asset and Liability Management (ALM) models under uncertainty were undertaken by (Kallberg et al., 1982) and (Kusy & Ziemba, 1986). Subsequently a number of substantial applications were developed, for banks, hedge and mutual funds, insurance companies and wealthy individuals. Quoting a few applications, it is worth considering the Computer-aided asset/liability management (CALM) model (Consigli & Dempster, 1998), a multistage stochastic pro-

`

gramming model which has been applied with good results to various environments. An overview of models applied to individual asset and liability management, as well as hedge funds is given in (Ziemba, 2003). Various paradigms applied to ALM are compared in (Schwaiger, 2009), which evaluates deterministic, SP, Chance Constrained, Integrated Chance Constrained models applied to a pension fund. In portfolio selection problems, risk measures, and in particular coherent risk measures (Artzner et al., 1999) have become increasingly widespread, and applications of stochastic dominance (SSD) to portfolio selection is lately being explored (Dentcheva & Ruszczynski, 2006), (Roman et al., 2006), (Fábián et al., 2009).

**Supply chain:** Supply chain has been historically a fertile field of application for stochastic programming, as the uncertainty in the products demand, costs and supply can hinder the quality of solutions obtained through deterministic modelling. Large scale applications are found in (Escudero et al., 1999), (Koutsoukis et al., 2000), (Alonso-Ayuso et al., 2003), (MirHassani et al., 2000). A comprehensive literature review is given in (Stadtler, 2005)

**Transportation:** Transportation and logistics were some of the earliest applications of stochastic programming. Fleet management in airline operations was used in (Dantzig, 1955) to motivate the need of stochastic programming. A stochastic formulation of the empty car distribution problem in the railroad industry is given in (Jordan & Törnquist, 1983). A review of such models and algorithms is given in (Törnquist, 2005). Aircraft scheduling is another area of growing application, although alternative modelling approaches have been successfully applied (Yu & Li, 2000), (Beasley et al., 2001), stochastic programming has been used too; see (Yen & Birge, 2006) for a model and a solution algorithm.

**Telecommunication:** Telecommunications has a long tradition of application of advanced mathematical modelling methods. While the traditional design approach is centred on minimization of the network costs under technological and quality of service constraints, systematic application of stochastic programming techniques includes incorporation of modern tools like evaluation of real options. Comprehensive models, which include pricing and strategic decisions, provide a motivation for

`

further development of this methodology. For examples of use of different kind of SP models (SLP, SMIP, SSOCP) in network design see (Gaivoronski, 2006), (Ntaimo & Sen, 2005) and (Maggioni et al., 2009).

**Energy:** Since most energy investments or operations are faced with uncertainty, a stochastic programming approach is normally meaningful. Many of the SP models deal with power generation investments, but also oil and gas applications have a relevant role in energy optimisation problems; see among others (Maggioni et al., 2007). (Escudero et al., 1998) propose several models about long-term planning of electricity and energy generation. Hydro Thermal optimization problems examples are (Sen & Kothari, 1998), (Nowak et al., 2000), (Escudero & Monge, 2008). With the liberalization of the energy markets, considering alternative investments and energy spot prices has become common in scheduling energy production, see (Fleten et al., 2009), (Konig et al., 2007).

## 1.5 Scenario Generation

*Scenario Generation* is the term normally used to describe the process of creating a tree structure and associated discrete scenarios which are used to describe the uncertain parameters in SP models. The uncertainty representation by scenarios can be summarized as a four steps process, of which the first two are required at modelling time, the latter two at run-time:

1. **Model the uncertainties with (discrete) random processes (or distributions for single stage SP)**

   The modeller is here required to write his assumptions about the uncertainty in mathematical form; the outcome of this step is a random process or a probability distribution. As discussed in section 1.2 and in (Birge & Louveaux, 1997), these analytic models are not suitable for direct use in SP problems.

2. **Approximate (discretize in case of continuous random processes or aggregate in case of discrete ones) the chosen random processes with a tree of discrete scenarios**

   A range of techniques can be used in this step, which approximates the output of the model defined in step 1 with a scenario tree. There is "*both a science and*

`

*an art*" (Casey & Sen, 2005) to this process, and a balance between fine discretization (that lead to numerically unsolvable problems) and coarse discretization (that could overlook important realizations) has to be obtained. Two related approaches can be identified, one is based on statistical approximations (as in (Høyland & Wallace, 2001)) and the other on approximation theory (as in (Hochreiter & Pflug, 2007), (Pflug, 2001), (Dupačová et al., 2000)).

3. **Estimate the parameters for the model of randomness**

   Given some data about the reality (usually, this data set comprises historical observations) the modeller then is required to estimate the uncertain parameters for a model of randomness (i.e. for a normal distribution, mean and variance).

4. **Generate the scenario tree**

   A scenario tree is then generated by applying the model/algorithm crafted in steps 1 and 2, and the data provided by step 3. This scenario tree is then introduced in the description of the SP programming model which in turn is processed by an SP solver.

The role of a scenario generator is very important in describing an SP problem. This thesis is mainly concerned with modelling aspects of SP and within the scope of this thesis, the concept of *Scenario Generators library* is introduced, which is a collection of models of randomness which have been produced by steps 1 and 2 of the modus operandi above. In the context of decision making, a scenario generator captures in a procedural form a domain-specific model of randomness. Operatively, the problem owner may choose between various methods which are part of the *SG library* to model the uncertainty at hand, evaluate its performance with the current decision model (i.e. *stability tests*, see paragraph 2.3) and use it to obtain the *ex-ante* decision (see (Di Domenica et al., 2009)). The problem owner can then evaluate the decision obtained against real data (back testing, or stress testing) or against realizations obtained using a different scenario generator (see 0). A short and not comprehensive list of scenario generators, their applications fields and some references is summarized in Table 8 below .

`

| Modelling Paradigm | SG Method | Origin | Application Field | Reference |
|---|---|---|---|---|
| Econometric Models and Time Series | AR (p) | Autoregressive Models and Generation of Data Trajectories. | Finance, Supply chains, Environment models | (Box et al., 1976) |
| | MA (q) | Moving Average Models and Generation of Data Trajectories | Finance, Supply chains, Environment models | (Box et al., 1976) |
| | ARMA (p,q) | Autoregressive Moving Average Models and Generation of Data Trajectories. | Finance, Supply chains, Environment models | (Box et al., 1976) |
| | GARCH | Generalised Autoregressive Conditional Heteroscedasticity and Generation of Data Trajectories. | Finance, Supply chains, Environment models | (Bollerslev, 1986), (Engle, 1982) |
| | VAR | Vector Auto Regressive Models and Generation of Data Trajectories | Finance, Supply chains, Environment models | (Fair & Shiller, 1990) |
| | BVAR | Bayesian Vector Auto Regressive Models and Generation of Data Trajectories. | Finance, Supply chains, Environment models | (Ansley & Kohn, 1986) |
| | Reduced Rank Regression | Generation of Data Trajectories | Finance, Supply chains, Environment models | (Engle & Granger, 1987) |
| **Modelling Paradigm** | **SG Method** | **Origin** | **Application Field** | **Reference** |
| Geometric Brownian Motion | Wiener Processes | Brownian Motion and Diffusion Processes | Finance and Environment models | (Freedman, 1972) |
| | Generalised Wiener Processes | Brownian Motion with drift and Diffusion Processes | Finance and Environment models | (Bollerslev, 1986) |
| Artificial Intelligence | Neural Gas | Neural Networks. | Supply Chains, Energy, Environment models | (Martinetz & Schulten, 1991) (Fritzke, 1995) |
| Statistical Approaches | Property Matching | Statistical Approximation | Supply Chains, Energy models | (Høyland & Wallace, 2001) |
| | Moment Matching | Moment Fitting | Supply Chains, Energy, Environment models | (Høyland et al., 2003) |
| | Non Parametric Methods | Discretisation | Finance and Environment models | (Høyland & Wallace, 2001) |

`

| | | | | |
|---|---|---|---|---|
| | SG Forecasting Methods | Quantile Regression and Forecasting Methods | Finance, Supply chains, Environment models | (Tomasgard et al., 1998) |
| Sampling | Random Sampling | Discrete Sampling | Finance and Environment models | (Jobst & Zenios, 2003) (Jobst et al., 2006) |
| | Stratified Sampling | Interval Sampling | Finance and Environment models | (Jobst & Zenios, 2003) (Jobst et al., 2006) |
| | Bootstrap | Discrete Sampling | Finance models | (Efron, 1979) (Efron & Tibshirani, 1997) |
| | Monte Carlo | Sampling | Finance models | (Jerrum & Sinclair, 1997) |
| | Markov Chains | Probability Interval Sampling | Finance, Energy, Supply Chains, Environment | (Jerrum & Sinclair, 1997) |
| | VECM | Random path and Vector Error correction | Finance | (Volosov et al., 2005) |

**Table 8 List of SG methods and applications**

## 1.6 An architecture for an SP modelling system

The acceptance of a modelling paradigm depends heavily on the availability of appropriate research results and tools that are based on such research and support modelling functionalities. The field of Stochastic Programming has a very active research activity that comprises various fields (from stochastic processes and models of randomness to solution algorithms) but the lack of tools that are designed specifically to support it has somewhat slowed down its adoption by analysts and OR practitioners. There have been efforts in this direction, among which one of the earliest is the precursor to this work (see (Valente, 2002)), and in recent years there has been considerable progress.

What is still lacking is a tool that can exploit, through careful categorisation and design of data structures and classes, concepts that come from the research in the field like compact instance representation and decomposition methods and that allows an easy implementation of SP models.

`

In this thesis I report on the design of software tools for Stochastic Programming, and I present and formalize the knowledge that I have acquired in recent years during which period the SP software system SPInE (Valente et al., 2001), (Valente et al., 2002-2011) was completely redesigned and implemented.

`

A high level conceptual outline of a SP modelling system from a computer science perspective is given in Figure 7 below:

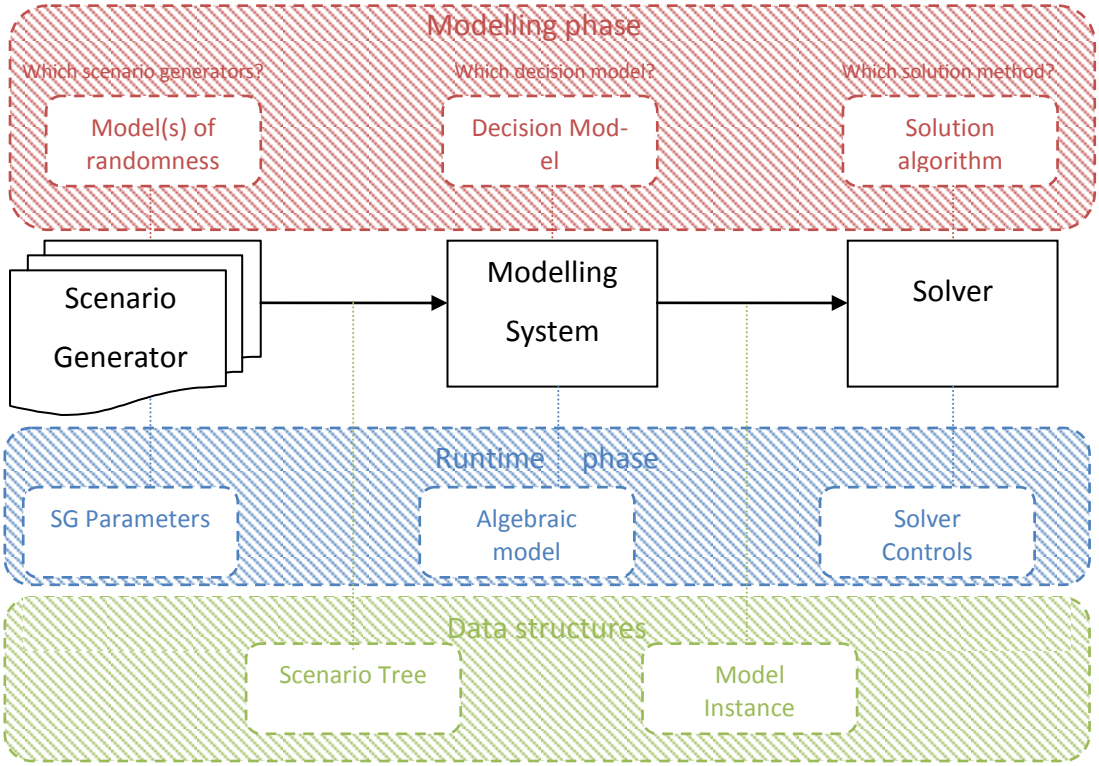



**Figure 7 High level overview of a SP modelling system**

In Figure 7 the three main modules of an SP modelling system (in black) are set out in the middle; these modules highlight three different aspects of the process of solving an SP problem. The other modules in this figure, displayed in different colours, correspond to different operative phases of the modelling process; these phases are described below:

- **Modelling Phase** (in red): the "strategic" steps the modeller has to follow to successfully set up the system to solve the problem, namely: formulate the decision problem, determine and model the randomness (decide what SG to use) and decide what solution algorithm to invoke
- **Runtime Phase** (in blue): to ultimately obtain results from all the knowledge formulated and formalized in the previous step, the consumer of the model needs then to: fit the models of randomness chosen at the modelling phase to the problem instance under examination, formulate the decision model in

`

a format which the software can interpret and eventually *tune* the solver or solution method for use

1. **Data Structures** (In green): describes how data instances are captured and gives an insight into the information that is passed between the modules. We refer the reader to section 3.1 for a more detailed discussion about the model instance format, and to 1.2 for an analysis of the scenario tree structure.

The high level schema reported in Figure 7 is expanded in the following chapters of this thesis. An important aspect is the control structure that is needed for setting up a simulation and investigation framework; this aspect is considered in detail in 0.

## 1.7 Outline of the thesis

In this thesis we provide a general review of the state-of-the-art of software tools which support stochastic programming modelling. In this introduction, we have given an overview of the various software components that constitute an integrated software tool for SP. In following chapters of this thesis we focus on three different stages of the modelling and solution process. The thesis is structured as follows: Chapter two provides an overview of the software tools available for SP, examining in particular the existing modelling languages. A model is then explicitly implemented using the modelling languages AMPL, using the deterministic equivalent formulations, and SAMPL, exploiting the advanced features of the language. In Chapter three we focus on the solution stage, illustrating requirements and characteristics for an SP solver. Instance level formats are described, which allow the communication between the modelling systems and the solvers; sections 3.2 and 3.3 give an overview of the deterministic equivalent and decomposition techniques used to solve various classes of SP problems, and some benchmarks and considerations on their relative performances are made in section 3.4. Finally, in section 3.5, the architecture of such solver is introduced, with the novel concept of automated mapping between model classes and solution techniques that the author believes is central to the development of a usable and performing SP integrated modelling system. Scenario Generation is introduced in Chapter 4. A brief overview of the scenar-

`

io generation process and its place in the modelling process is given in section 4.1. Some common methods and their application areas are given in section 4.2. Some desirable properties of scenario generators are introduced in section 4.3. The necessary step of abstraction is then described in section 4.4, abstraction that then leads to the concept of a scenario generation library, which is then described in both from the programmer's (0) and the modeller's (4.6) point of view. Chapter five introduces the concept of workflows, and a rationale on their use to create an extensible investigation framework. The activities (or atomic operations) composing such a workflow are presented in section 5.3 and some sample cases are given in section 5.4. Finally, summary, conclusions and future works are presented in Chapter six.

`

# Chapter 2 Software tools for Stochastic Programming

This chapter introduces the conceptual components of an integrated tool for stochastic programming, and provides a broad overview of the state-of-the-art technologies and software available nowadays. Section 2.1 decomposes the process of using Stochastic Programming for Decision Making into various research problems and links each area to the software tools and components that help in the stages of the process. Moreover, it sheds some light on the logical entities (interfaces and data structures) that are used along the process, and links these entities to the appropriate section of this thesis.

Sections 2.2 to 2.4 can be considered literature review: one of the first topics to face when getting acquainted to software tools for SP is algebraic modelling languages (AMLs), which are presented in Section 2.2 in general, while section 2.3 focuses on the current development of SP-focused languages. In section 2.4 we make the case for alternative ways of representing the problems in a modeller's perspective.

In Section 2.5 we present a brief overview and introduction to a modelling language (AMPL) and its extended version (which we call SAMPL), implementing in both languages the example given in section 1.2 and showing the benefits of the formulation using SAMPL.

The author contribution lies in the extensive design studies to identify the requirements of a stochastic modelling system, and in its implementation. What has been inherited from his predecessors and collaborators has been almost completely rewritten and is now a working application used by many researchers. In addition, the language extensions for CCP, ICCP and Robust Optimization have been completely designed by the author.

`

## 2.1 An Information Technology framework for SP

Progresses in Stochastic Programming involve and require research in various fields, among which we identify (see Figure 8 below):

- **Modelling (applications)** which can be further divided in:
  - **(a)** Decision models – the mathematical description of the decision processes using *modelling paradigms as SP, MSP, CCP, ICCP, RO* (see sections 1.2 and 2.2)
  - **(b)** Model of randomness – the mathematical representation of the uncertainty involved, or *Scenario Generation* (see section 4.1)
- **Model representation formats**
  - **(c)** Structural level – how to communicate the decision model structure to a computer, independently from the particular data on which the model is instantiated (see sections 2.2 and 2.3)
  - **(d)** Instance level – how to represent the model generated from the combination of the structure and the data (see sections Deterministic Equivalents for SP, CC, ICC, Robust Optimization models1.3 and 2.4)
- **Solution methods**
  - **(e)** Deterministic equivalents – traditional LP/QP/QMIP solution techniques (Simplex, Interior point method, branch and bound) can be applied to the deterministic equivalent formulation (see sections 1.3 and 3.2)
  - **(f)** Decomposition methods – specialized algorithms can be applied if the problem structure is maintained and communicated to the solver (see sections 2.4 and 3.3)
- **(g) Results evaluation and interpretation** – the results of the optimisation should be evaluated and tested, to check if the assumptions about uncertainty and decision process were adequate (see 0)

All these fields are intertwined and the outcome of these researches should be taken into consideration when designing an application to support Stochastic Pro-

`

gramming. The importance of standardized software tools is obvious from the current market, in which the emergence of a standard usually makes the difference between a successful paradigm and an ignored one.

Another view of Figure 7 is the following, this time highlighting the research areas involved in the various components of the software system:

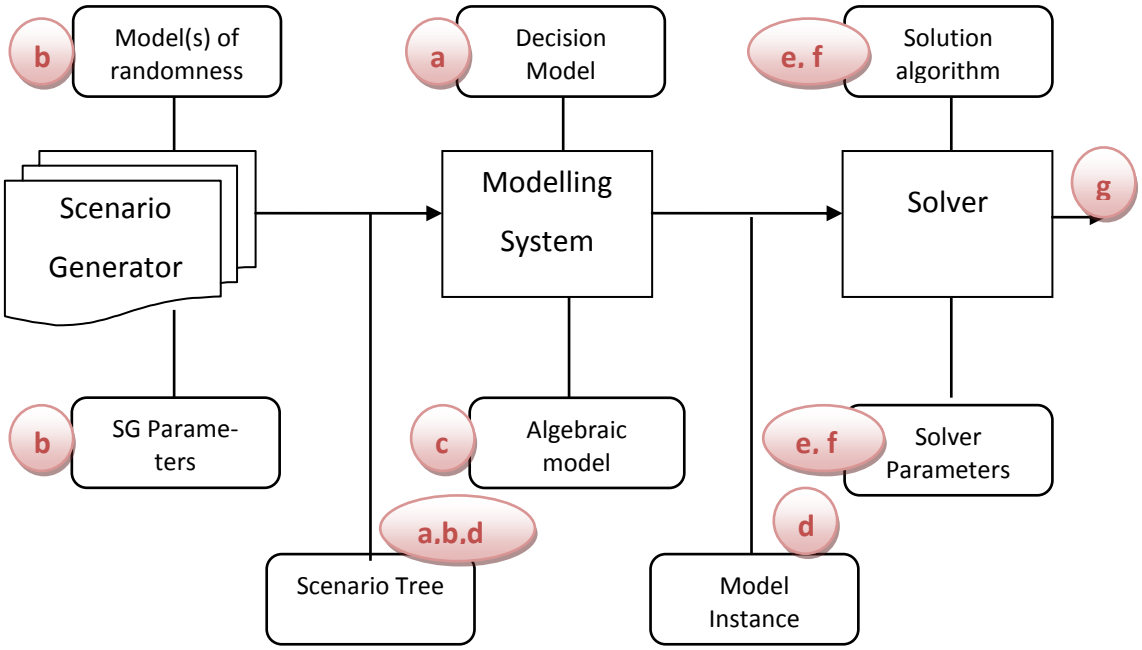



**Figure 8 Research areas vs software components**

The combination of Figure 8 and this section gives an ordered overview of the contents of this thesis in respect to SP modelling paradigm. Exploded versions of some of the blocks are presented in the corresponding sections, along with a more thorough description of each one of them.

## 2.2 Algebraic Modelling Languages

When mathematical programming was first introduced, models were generated by ad-hoc computer code written using procedural languages such as FORTRAN. These programs were used to generate the matrix related to a given LP/IP model; an appropriate solution algorithm was used to process the matrix and to find and report the optimal solutions. This approach was neither scalable nor elastic: even minor changes in the model's data or structure require major adjustments of the genera-

`

tion code. To aid practitioners in the creation of mathematical programming models, new type of languages appeared during the 1970's, called *matrix generators*. These languages enabled the generation of an LP/IP model's matrix in a specific format, which became a standard interface between modelling and solving systems. The format, namely *Mathematical Programming System* (MPS), was introduced by IBM (IBM World Trade Corporation, 1976) and, with some minor modifications, is still used today. Amongst the main matrix generators were OMNI (Hareveley Systems, 1976), DATAFORM (Ketron, 1975), MRGW (IBM World Trade Corporation, 1977), GAMMA (Sperry Univac Computer Systems, 1977) and MGG/RWG (Scicon Computer Services, 1975) . These languages, however, were affected by a number of limitations; in particular, the formulation was data dependent, thus the model could not be easily re-instantiated using different data sets.

Modern modelling systems for mathematical programming are based on *algebraic modelling languages* (AMLs), which enable the definition of models via symbolic algebraic expressions. Algebraic modelling systems (that is, software systems which support algebraic modelling languages), interpret the algebraic model and use a given set of data to create model instances in a MPS format or equivalent. The availability of algebraic modelling languages has contributed to the acceptance of mathematical techniques in the following ways:

1. Model development and prototyping has become a high productivity process. This has led to widespread acceptance of optimisation by the end user community based on the proof of concept application rapidly developed by OR/MS analysts
2. Many examples of integration of optimisation techniques into Decision Support Systems (DSS) can be found used in real world applications

Algebraic modelling languages are traditionally declarative languages; using this class of languages, the modeller describes *what* a problem is, without specifying *how* the problem is to be solved. The algebraic notation used in the formulation of MP models and supported by the AMLs plays an important role in the comprehension and maintenance of the models (Kuip, 1993). One of the most important ad-

`

vantages is that the algebraic formulation implies the *abstraction* of the model from the specific *instance* of the problem, thus enabling the separation between data modelling and modelling of the problem's structure. See (Fourer, 1997) for a more thorough discussion of the issues of modelling data in relation to MP models. MPL (Maxmimal software, 2002), LINGO (Lindo Systems Inc, 2008), CAMS (Lucas & Mitra, 1988) are some representative algebraic modelling languages which are purely declarative.

In more recent times, some AMLs have introduced procedural features, such as IF-THEN-ELSE statements and looping constructs: UIMP (Ellison & Mitra, 1982), LPL (Hürlimann, 1993), AMPL (Fourer et al., 2002), GAMS (Brooke et al., 2008), OPL (Van Hentenryck et al., 1999) and AIMMS (Bisschop & Entriken, 2009) belong to this family of mixed declarative/procedural languages. Procedural constructs enable a closer coupling of modelling systems and solvers, which can be exploited for techniques such as column generation and the implementation of decomposition algorithms (see section 3.3).

Algebraic modelling languages enable the formulation of mathematical programming models in terms of entities such as *sets* and *indices, parameters, decision variables, constraints* and *objective functions.* The relationships between these entities are described in (Dominguez-Ballesteros et al., 2002) and are shown in Figure 9, adapted to the new developments in the field.

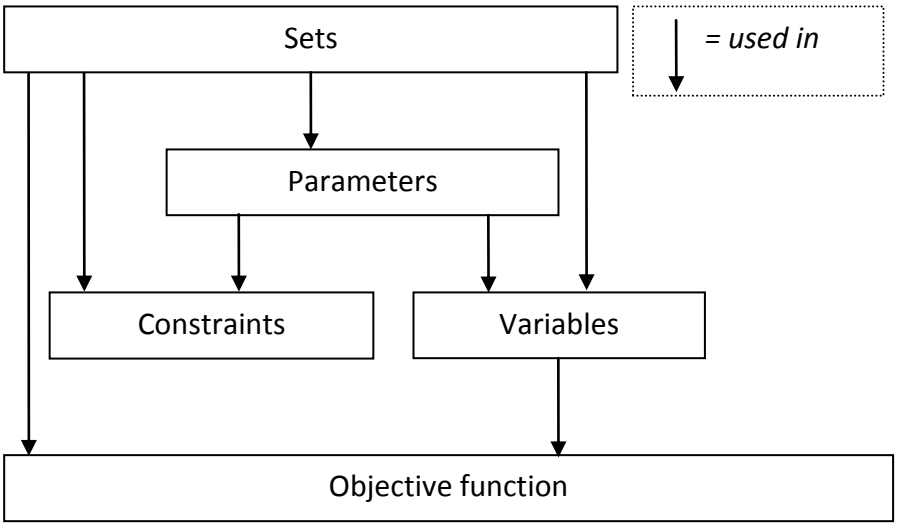



**Figure 9 Components in an algebraic model**

`

An important feature of algebraic modelling languages is the support of sets and indexing techniques. Indexing is derived from ordinary algebra and enables mathematical abstraction; indices and sets play an important role in the modelling of large scale problems, as these facilitate the classification of entities of conceptually similar nature. Algebraic modelling languages support several types of sets, including ordered sets, derived sets and hierarchical sets. Most of the ALMs also support fundamental sets operations such as *union, difference, intersection, Cartesian product, selection* and *join.*

Some AMLs such as AIMMS and LPL are capable of performing unit consistency checks. This is a valuable feature since one of the most frequent errors in the formulation of MP models is caused by the use of inconsistent measurement units in the algebraic expressions (Bisschop, 1986). The ability of expressing logical conditions is another important feature of the AMLs. Williams shows in (Williams, 1987) how logical conditions can be introduced into a mathematical programming model by mean of binary variables. In (Mitra et al., 1994), the authors propose an extension of the MPL modelling language which enables the automatic translation of logical relations such as *and, or,* $\Rightarrow$ and $\Leftrightarrow$ into MIP models. Fourer (Fourer, 1998) also discusses this issue and suggest the use of *logical operators, conditional operators* and *variables in subscript* in the AMPL language.

## 2.3 Extensions of Modelling Languages for SP

It is always possible to formulate a SP problem using algebraic modelling languages which implement the constructs presented in the previous section, however, doing so, leads the problem owner to difficulties of two kinds:

*Modelling issues:* the formulation of the SP problem becomes unnatural, as nonanticipativity constraints as well as the deterministic equivalent formulations of ICCs and CCs (see Section 1.3) must be explicitly implemented, thus distracting the modeller from the decision problem at hand. Moreover, the three classes of problems (Wait and See, Expected Value and Here and Now, see Appendix A) along with the related stochastic measures (Value of Stochastic Solution ad Expectation of the

`

Expected Value, see Appendix A), if of interest, must be separately formulated and solved, even though they are conceptually part of the same family of models and constitute the basis for computing stochastic measures.

*Computational issues:* when formulated as deterministic equivalents, the resulting matrix dimension increases linearly with the number of scenarios and exponentially with the number of stages (assuming a tree shape with a constant number of branches at each stage) and the solution time grows steeply with it. Moreover, the memory occupied by the matrix could become a problem for very large problems when formulated as deterministic equivalent, but this could be overcome avoiding the replication of *deterministic* data for all scenarios, which is possible if the system keeps the information regarding which parameters are stochastic and which are not. This kind of information allows the solution of the problem by means of decomposition techniques, which are characterized by a much better scale-up property in respect to the deterministic equivalent solved by conventional means (see Section 3.3).

These modelling and solving issues can be to a large extent alleviated using a modelling system (and therefore a modelling language) which is specialized for stochastic programming problems. There have been a few attempts to do so, and they are briefly listed in this section.

Gassmann and Ireland in (Gassmann & Ireland, 1995) address the problem of defining scenario-based recourse problems using existing AMPL constructs. Scenarios are specified parametrically and the scenario data can either be imported or ideally computed by the AMPL modelling system. The scenario tree structure is represented by first defining a base scenario, and additional scenarios sharing at least the root node with the base scenario are characterised by a parent scenario and the first stage in which the scenario differs from its parent.

Fourer in (Fourer, 1996) proposes *extensions* to the AMPL modelling language. These permit the definition of a stochastic programming problem with recourse in terms of a multistage (deterministic) model, a tree of data scenarios for the model, and a stochastic framework to specify the stages and optionally the scenarios and

`

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

In (Gassmann & Ireland, 1996) there is a proposal for other extension to the AMPL modelling language, mainly for the definition of probability distributions of the random parameters. Again, a language construct `random` is introduced in distribution-based recourse problems to identify the random parameters and the variables that depend on them.

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

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

`

The authors of (Fourer & Lopes, 2009) propose an extension of the AMPL modelling language whereby stochastic models are formulated using a representation based on dynamic programming.

AIMMS provides functionalities to express SP programs with recourse, where the user can define its own routines to generate the values of the random parameters. The model can then be generated as deterministic equivalent or solved by a Bender's decomposition based algorithm embedded in the system (Bisschop & Entriken, 2009).

In (Colombo et al., 2009) the authors describe their SML (Structure-Conveying Modelling Language), which is implemented as language extensions to AMPL. It extends AMPL with object-oriented features that allow the users to construct models as a combination of sub-models while preserving the block structure, so that the structure can be passed to a solver and exploited. It is not specifically designed for SP, permitting the formulation of any model that exhibit a block structure.

Microsoft Solver Foundation (MSF) offers now the possibility to express stochastic programming problems in their own modelling language called OML or in any .NET language but the support for SP is still at an embryonic stage and not much documentation has been released at the moment of writing (Microsoft Corporation, 2010).

The software system called SPInE (Stochastic Programming Integrated Environment) has been implemented by the research team in CARISMA of which the author is a member, it supports and interprets the language SAMPL (Valente et al., 2009). This is again an extension to the AMPL language specifically designed to support the formulation, generation and solution of various classes of SP problems. In this thesis we highlight the author's contribution to the design and implementation of such system.

### *Analysis of modelling issues*

The difficulties that arise when using non-specialized modelling languages to formulate SP problems are mainly due to the lack of constructs for the definition of the

`

randomness of the model coefficients and the scenario tree structure. A stochastic programming model can be considered as a linear programming model extended and refined by the introduction of random parameters (see Figure 10). More precisely, the underlying LP optimisation model is extended by taking into account the probability distribution of the model's random parameters. Such distributions are provided by the models of randomness used in *scenario generators* (see 0), which are specific to the particular optimisation problems under investigation.

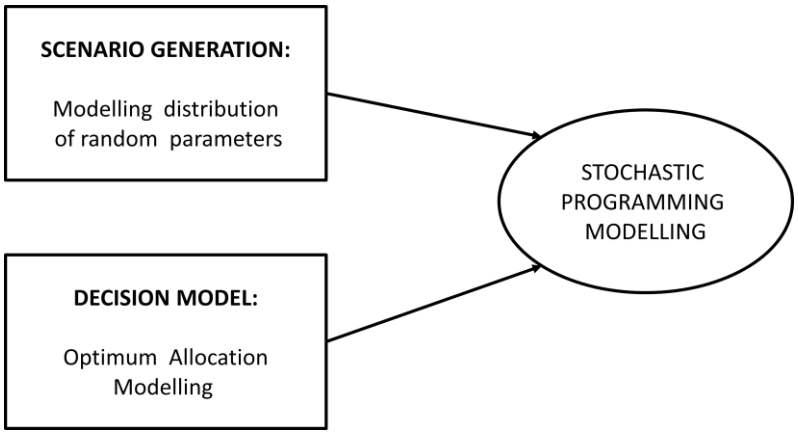



**Figure 10 Combined Paradigm**

In general, different categories of stochastic programming problems require different language features to express the random nature of the problem. We use the term *stochastic framework* to denote the information represented by these constructs.

### *Modelling scenario-based recourse problems*

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

`

marise, a stochastic framework for scenario-based recourse problems requires constructs for the definition of stages, scenarios and random parameters (see Table 9).

| Entities | Language Requirements |
|---|---|
| **Stage information** | assignment of variables and constraints to stages |
| **Scenario information** | scenario set<br>tree structure<br>scenario probabilities |
| **Random parameters** | declaration of the random parameters in terms of the scenario set |

**Table 9 Language requirements for scenario based recourse problems**

### *Modelling distribution-based recourse problems*

A number of researchers have proposed extensions to algebraic languages for the formulation of this class of problems (Fourer, 1996), (Gassmann & Ireland, 1996), (Fourer, 2001), (Gay, 2001). In this work, we focus on the class of scenario-based recourse problems and only outline the requirements for distribution-based models. Distribution-based recourse problems rely on the declaration of discrete or continuous probability distributions for the random parameters. If all random parameters are characterised by discrete distributions, the scenario tree is implied by the joint realisations of the random parameters. If one or more distributions are continuous, then there are infinitely many possible outcomes for the random parameters and the tree structure must be sampled from the joint distributions. A stochastic framework for distribution-based problems requires constructs shown in Table 10.

| Entities | Language Requirements |
|---|---|
| **Stage information** | assignment of variables and constraints to stages |
| **Random parameters** | declaration of the probability distributions associated with the random parameters |

**Table 10 Language requirements for distribution based SP problem**

`

### *Overview of our approach*

We present methods of extending algebraic modelling languages based on the concepts of *underlying deterministic model* and *stochastic framework*. The underlying deterministic model is formulated using the standard constructs provided by an AML. Using several new constructs, the modeller then declares the stochastic framework, which links the underlying deterministic model to the model of randomness. More specifically, the underlying deterministic model represents a family of independent (wait-and-see) models, while the stochastic framework imposes the nonanticipativity implied by the structure of the scenario tree. The resulting model is the formulation of the stochastic programming recourse problem.

### *The underlying deterministic model*

In the formulation and investigation of a stochastic programming problem, it becomes necessary to identify the underlying deterministic model (also called the *core* model). This can be the expected value problem or a problem corresponding to any sample path of the scenario tree. The underlying deterministic model captures the logical structure of the problem as well as the dynamical relations among decision variables, their bounds, and the objective function. The Stochastic Programming research team in CARISMA has, in the design of the language SAMPL and in the implementation of its interpreter SPInE (Valente et al., 2001), (Valente et al., 2009), (Valente et al., 2002-2011), followed the approach of constructing the core model to be parametric in the dimension of scenarios. All variables and constraints are indexed over the scenarios, which are the elements of a special set declared in the stochastic framework, and the objective function is the expected value over all scenarios of the individual objectives.

### *Declaration of the stochastic framework*

The stochastic framework depends on the type of stochastic programming model that is being developed. For instance, scenario-based recourse problems require the explicit declaration of the scenario tree structure while, in a distribution-based

`

recourse problem, the AML should provide a set of constructs for the definition of probability distributions.

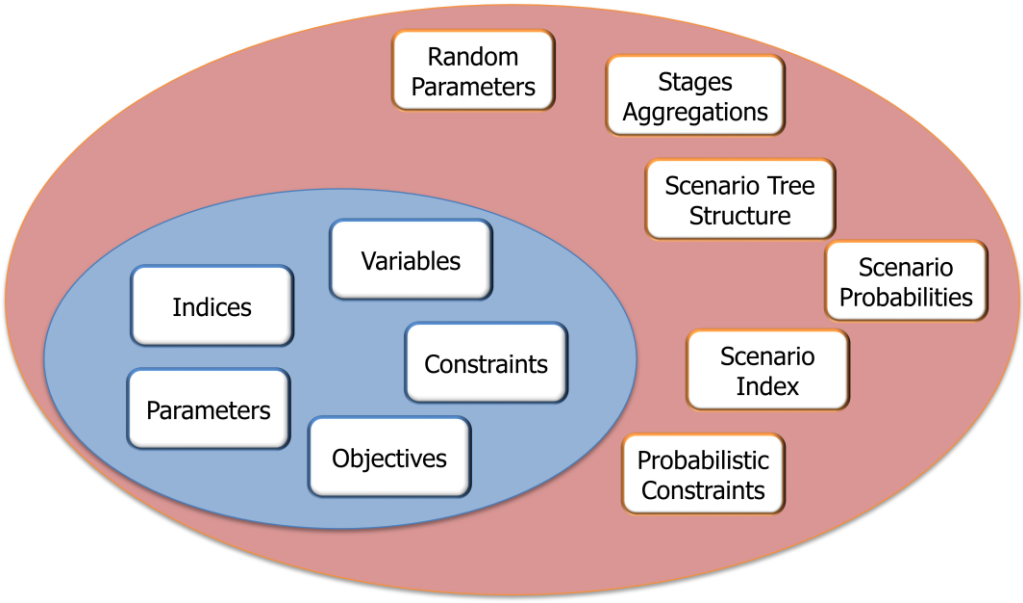



**Figure 11 Extended constructs for SP**

Figure 11 shows how the basic constructs of a modelling language for linear programming are extended to capture the stochastic framework. The definition of the new constructs is adapted to be consistent with the grammar of the underlying modelling language. We have successfully applied this approach to the AMPL language, but the same ideas can be adapted to virtually any other AML; see for instance (Valente, 2002). The syntax of the extended language constructs for stochastic AMPL (which we call SAMPL) is defined in section 2.4; the modelling system that interprets SAMPL and generates the instance of the model is called SPInE (Stochastic Programming Integrated Environment).

## 2.4 Alternative representations

As repeatedly observed in this thesis, the deterministic equivalent formulations of SP problems suffer of some disadvantages in respect to their "direct" representations. From the modeller's point of view, he is forced to change the model in accordance with the structure (event tree) he wants to enforce through the use of non-anticipativity constraints, or to add variables and constraints to the problem to

`

capture (integrated) chance constraints, and robust optimisation problems; this is an added complication as shown in section 1.3. The non-preservation of the problem's structure and the redundancy of the instance-level representations derived from the use of DEQs are other important issues which are discussed in section 3.1.

This set of reasons makes a strong case for a language which supports direct representation of SP problems, thus maintaining the meta-information which defines the SP framework and allowing smart interaction with solvers and scenario generators. The author has contributed to this research by designing, implementing and extension such a system, which is an integrated software system for the representation, generation and solution of SP problems. The AMPL modelling language (Fourer et al., 2002) has been modified with subsequent extensions (Valente, 2002), (Valente et al., 2009), (Valente et al., 2011) to directly represent an increasingly large class of problems and features – among which, still unpublished, are the extensions for robust optimization problems and for scenario generation – and the underlying system SPInE (Messina & Mitra, 1997), (Valente et al., 2001) re-implemented and partially redesigned to support seamless (language supported) connection to scenario generators, parallel model generation and a revised interface with the solvers.

## 2.5 Modelling languages perspective: a tour on AMPL and SAMPL

The aim of this section is twofold: firstly, it presents the modelling language constructs interpreted by the system developed by the author, secondly, it shows the difference in syntactic clarity between expressing the various classes of SP model using these language constructs and without.

It is presented as a tutorial-like description, using the personal planning model presented in section 1.2; the various model classes are presented in the same order as in above mentioned section. Each class is formulated both in AMPL and SAMPL (AMPL including the stochastic extensions), to illustrate the gain in compactness

`

and functionalities achieved using the extended syntax, the syntax is interpreted by the system developed by the author.

### 1) Personal Planning deterministic model in AMPL

The AMPL code for the model, is set out below. For a more detailed description of the features and the syntax of the language, the reader is referred to (Fourer et al., 2002).

```
set DUTIES;
set TIME;

param WakingHours;
param StressFactor;
param RelaxFactor;
param WorkRequired{DUTIES};
param Productivity{DUTIES, TIME};
param AverageWork >= 0, <=8;

var Work{DUTIES, TIME} >= 0, <= WakingHours;
var UnderWork{TIME} >=0;
var OverWork{TIME} >= 0;

minimize Stress: sum{t in TIME} (OverWork[t] * StressFactor -
UnderWork[t] * RelaxFactor);

subject to
DayLength{t in TIME}: sum{d in DUTIES} Work[d,t] <= WakingHours;
DeviationDefinition{t in TIME}: sum{d in DUTIES} Work[d,t] =
AverageWork + OverWork[t] - UnderWork[t];
TaskCompletion{d in DUTIES}: sum{t in TIME} (Productivity[d,t] *
Work[d,t]) = WorkRequired[d];
```

The data file, containing all the values needed for the model instance to be generated, is as follows:

```
set DUTIES := study work;
set TIME := 1 2 3;

param WakingHours := 12;
param StressFactor := 5;
param RelaxFactor := 4;
param WorkRequired :=  study 12   work 14;

param Productivity : 1       2      3 :=
study                0.9     1.2    1.3
work                 1       0.9    1.3;
```

`

## 2) Personal Planning SP multi stage

Introducing the scenarios for the productivity, as discussed in section 1.2, makes us firstly add some information to our data. A set (`SCENARIOS`) is added to represent all the scenarios, a parameter (`Prob`) contains the probability of each scenario and a parameter (`RProductivity`) represents the realizations of our scenarios.

```
param RProductivity
[*, *, 1] : 1       2       3 :=
study       0.9     1.44     1.56
work        1       1.08     1.56
[*, *, 2] : 1       2       3 :=
study       0.9     1.44    1.0504
work        1       1.08     1.0504
[*, *, 3] : 1       2       3 :=
study       0.9     0.96     1.56
work        1       0.72     1.56
[*, *, 4] : 1       2       3 :=
study       0.9     0.96     1.0504
work        1       0.72     1.0504;
```

### *AMPL Formulation (via deterministic equivalent)*

Following the explicit non-anticipativity representation presented in section 1.3, the multi stage model is expressed in AMPL as follows, where the lines added in respect to the deterministic version above are reported and all the changes to existing lines are highlighted in bold.

```
set SCENARIOS := 1..4;
param Prob{SCENARIOS} := 1/card(SCENARIOS);

param RProductivity{DUTIES, TIME, SCENARIOS};

var Work{DUTIES, TIME, SCENARIOS} >= 0;
var UnderWork{TIME, SCENARIOS} >=0;
var OverWork{TIME, SCENARIOS} >= 0;

minimize Stress: sum{t in TIME, s in SCENARIOS} Prob[s] *
(OverWork[t,s] * StressFactor - UnderWork[t,s] * RelaxFactor);

subject to
DayLength{t in TIME, s in SCENARIOS}: sum{d in DUTIES} Work[d,t,s]
<= WakingHours;

DeviationDefinition{t in TIME, s in SCENARIOS}: sum{d in DUTIES}
Work[d,t,s] = AverageWork + OverWork[t,s] - UnderWork[t,s];

TaskCompletion{d in DUTIES, s in SCENARIOS}:
```

`

```
sum{t in TIME} (RProductivity[d,t,s] * Work[d,t,s]) =
WorkRequired[t];
```

To complete the explicit non-anticipativity representation, a structure must be enforced to ensure that only the information available at each decision node influences the decision itself. We have therefore to add to the model the non-anticipativity restrictions, which depend on the tree structure of our choice (see section 1.2 for the mathematical formulation of the constraints below)

```
NAFIRSTSTAGE{d in DUTIES, s in SCENARIOS}:
Work[d,1,s] = Work[d,1,1];

NASECONDSTAGE1{d in DUTIES }:
Work[d,2,1] = Work[d,2,2];

NASECONDSTAGE2{d in DUTIES}:
Work[d,2,3] = Work[d,2,4];
```

### *SAMPL formulation*

The extended syntax SAMPL provides enables the modeller to capture the stochasticity in the model in a natural way, using ad-hoc constructs. Most importantly, the non-anticipativity constraints are not needed, as the generated model enforces the desired tree shape automatically. The modifications to the deterministic model to achieve its stochastic SAMPL formulation of the model are reported below; it should be noted that all Wait and See, Here and Now and Expected Value formulations can be generated automatically by the system starting from SAMPL formulation in SAMPL, and the stochastic measures VSS and EVPI are automatically calculated.

```
scenarioset SCENARIOS := 1..4;
probability Prob{SCENARIOS} := 1/card(SCENARIOS);
tree theTree := binary;
random param RProductivity{DUTIES, TIME, SCENARIOS};
var Work{DUTIES, t in TIME, SCENARIOS} >= 0, suffix stage t;
var UnderWork{t in TIME, SCENARIOS} >=0, suffix stage t;
var OverWork{t in TIME, SCENARIOS} >= 0, suffix stage t;
```

`

The compactness of this formulation in respect to the DEQ one is noticeable, and the shape of the tree is easily defined with the keyword **`binary`**, as the forecasted tree has this structure. For a full language reference, the reader is referred to (Valente et al., 2002-2011) .

### Scenario Generation

Although the random parameters can be generated by the procedure described in section 1.2, the data has to be explicitly provided to AMPL in order to be able to instantiate the model. With SAMPL, there is an alternative approach: the modeller can choose to program a scenario generator module (see Chapter 4) and the system will take care of generating the values at runtime.

**Using the developed libraries, accessible to anyone that wishes to extend the system with a new scenario generator, which contain the definition of some classes, the author has developed a scenario generator implementing the procedure described in C#. If the module follows the specified interface, the system is then able to see it.**

`

Figure 12 shows the application developed by the user discovering the scenario generator implemented for this model. The top window shows the dynamically populated list of available scenario generators (just one, in this instance) and its description and the parameters it needs. All this information is found out at runtime for compliant scenario generators. The user does not need to specify the realization explicitly; just the parameters needed to the scenario generator needs to be incorporated instead.

`

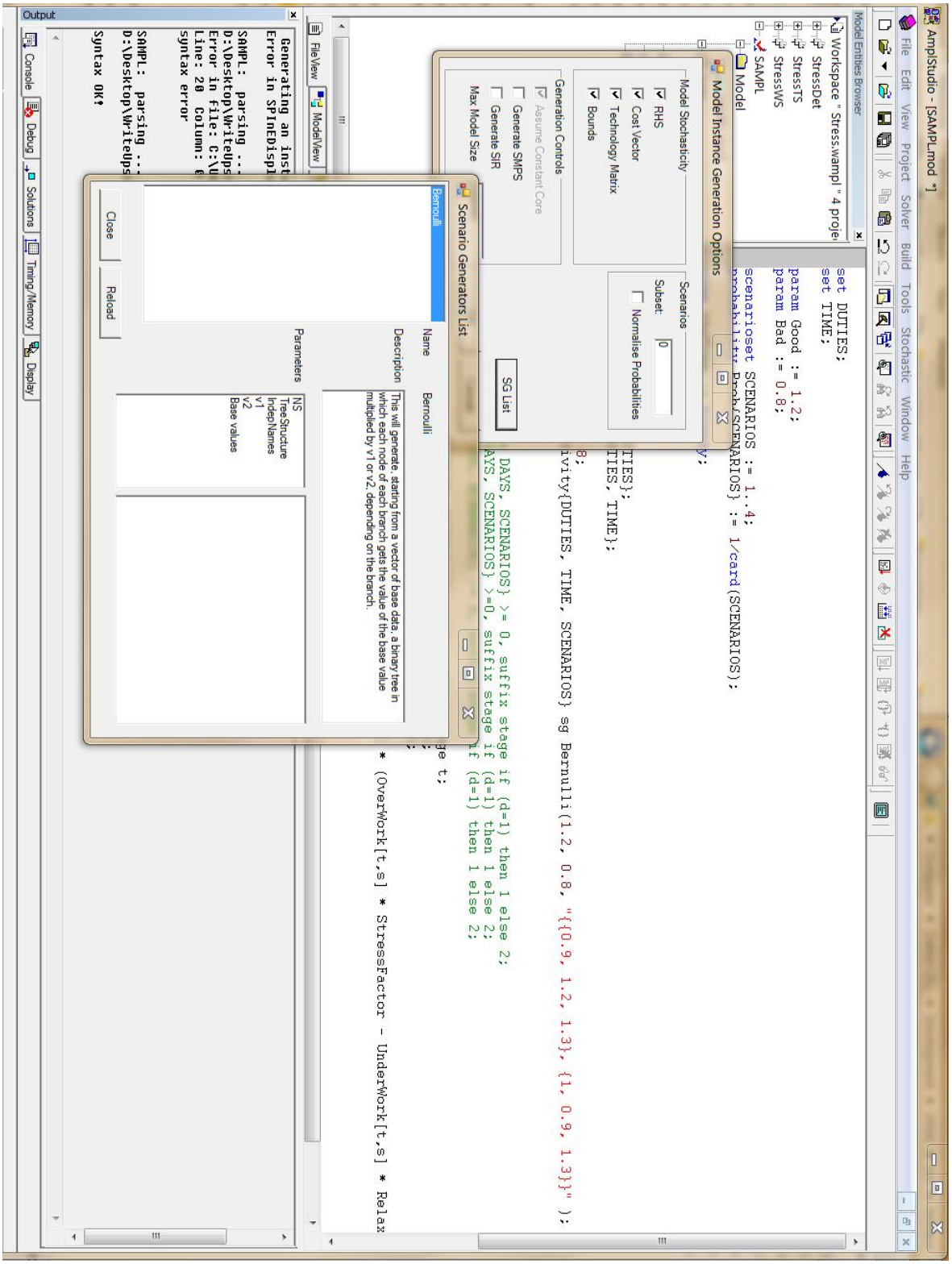



**Figure 12 Personal planning model: Screenshot from SG**

`

A C# code snippet of the generation procedure is reported below for completeness, it uses the classes `ScenarioTree` and `ScenarioTreeNode` which can be found in the SG library main module implemented by the author. It recursively populates a binary tree with arrays obtained multiplying each element of a base array by the numbers v1 or v2, depending on the branch. This is obviously implementing the scenario generation procedure intended for this model.

```
public void GenerateNode(ScenarioTree<double[]>.ScenarioTreeNode
node, double coefficient)
{
	node.Contents = new double[indepNames.Length];

	for(int i=0; i<indepNames.Length; i++)
	node.Contents[i]= baseValues[node.Stage][i] * coefficient;

	if(node.hasChildren)
	{
		GenerateNode(node.Children[0], v1);
		GenerateNode(node.Children[1], v2);
	}
}
```

At this point, the data can then be eliminated, and the declaration of the random parameter becomes:

```
random param RProductivity{DUTIES, TIME, SCENARIOS} sg Bernoul-
li(1.2, 0.8, "{{0.9, 1.2, 1.3}, {1, 0.9, 1.3}}" );
```

Two remarks can be made on this topic:

- Having the model expressed in this way, allows the modeller to increase the number of items in the set DUTIES, or increase the number of time periods in the model, without worrying to regenerate the data: it will automatically be generated by the system at runtime
- One limitation of the current system is apparent on this line: it is not allowed to pass AMPL parameters to the scenario generator. The solution to this problem is one of the future developments, which can be found in section 6.3.

`

### 3) Chance Constraints

#### *AMPL Formulation*

The AMPL formulation of the chance constrained version of the model follows once more the deterministic equivalent formulation. For this formulation, binary variables have to be added to count the amount of violations in the constraints, and a counting constraint has to be included. The deterministic equivalent formulation of the individual chance constraints expressed in 1.22 is the following (see section 1.3 for reference):

$$\sum_{t\in T}(W_{dts}P_{dts}) + WR_d\delta_{ds} = WR_d \quad \forall d \in D, \forall s \in Sc$$

$$\sum_{s\in Sc} Pr_s\delta_{ds} \leq \alpha \quad \forall d \in D \qquad \textbf{2.1}$$

$$where\ \delta_{ds} \in \{0,1\} \forall d, s \ , \alpha \geq 0$$

Translating that to AMPL is straight forward and it reads like:

```
param alpha := 0.6;
var delta{DUTIES, SCENARIOS} binary;

TaskCompletionCC{d in DUTIES, s in SCENARIOS}:
sum{t in TIME} (RProductivity[d,t,s] * Work[d,t,s]) +
delta[t,s]*WorkRequired[t] = WorkRequired[t];

CCCount{d in DUTIES}:
sum{s in SCENARIOS} Prob[s]*delta[d,s] <= alpha;
```

The artifices introduced in the model due to the deterministic equivalent formulation are highlighted in bold. It is worth noticing that this formulation introduces one binary variable for each scenario and for each individual constraint.

#### *SAMPL Formulation*

The formulation of the chance constraint using SAMPL extended syntax, as in (Valente et al., 2011) is:

```
AmountWorkedCC{d in DUTIES} {probability s in SCENARIOS:
sum{t in TIME}RProductivity[d,t,s] * Work[d,t,s] = WorkRequired[d]}
<= alpha;
```

`

Having a specific construct for this allows for a much cleaner formulation, the system can then check if a solver has a specific solution algorithm for this class of problems and eventually exploit it.

### 4) Integrated Chance Constraints

The deterministic equivalent formulation of the Integrated Chance Constrained model follows; this formulation limits the expected "units behind schedule" for each task to $\beta$. It includes the needed added variables and the added constraint calculating the expected violation.

$$\sum_{t \in T} (W_{dts} P_{dts}) + \mu_{ds} = WR_d \quad \forall d \in D, \forall s \in Sc$$

$$\sum_{s \in Sc} Pr_s \mu_{ds} \leq \beta \quad \forall d \in D \qquad \textbf{2.2}$$

$$where\ \mu_{ds} \geq 0\ \forall d, s\ , \beta \geq 0$$

The AMPL implementation of the formulation above follows:

```
param beta := 3;
var mu{DUTIES, SCENARIOS} >= 0;

TaskCompletionICC{d in DUTIES, s in SCENARIOS}:
sum{t in TIME} (RProductivity[d,t,s] * Work[d,t,s]) + mu[d,s] =
WorkRequired[t];

ICCAverage{d in DUTIES}:
sum{s in SCENARIOS} Prob[s]* mu[d,s] <= beta;
```

It is worth noticing that this formulation introduces one continuous variable for each scenario and for each individual constraint.

### *SAMPL Formulation*

The formulation of the integrated chance constraint using SAMPL extended syntax, as in (Valente et al., 2011) is:

```
ICCP{d in DUTIES}: expectation{s in SCENARIO} {WorkRequired[d] less
sum{t in TIME}RProductivity[d,t,s] * Work[d,t,s]} <= beta;
```

Once again, this reformulation is much more compact and readable and allows the system to use a solver that is especially designed to solve ICCPs through specialized algorithms (see (Haneveld & van der Vlerk, 2006) for an example).

`

## 5) Robust formulations

To avoid being too prolix on a topic – the formulation of robust optimisation problems – which is not central to this thesis, only the formulation given by Soyster (see section 1.2) is explicitly given here. Its only assumption is that we know the ranges in which the non-deterministic parameters can vary.

### *AMPL Formulation*

The deterministic equivalent formulation of the robust optimization problem was given in 1.34. The added parameters $\widetilde{P_d}$ are the amplitude of the allowed deviation from the mean value of the productivity values $P_{dt}$.

To formulate the robust optimization problem, we use the deterministic version as a starting point, then we add the following, which is the AMPL equivalent of 1.34:

```
var y[DUTIES, TIME] >= 0;

param ProdRange[TIME];

TaskCompletion{d in DUTIES}:
sum{t in TIME} (Productivity[d, t] * Work[d,t])
- sum{t in TIME} (y[d,t] * ProdRange[t])
>= WorkRequired[d];

YConstraint{d in DUTIES, t in TIME}:
-y[d,t] <= Work[d,t] <= y[d,t];
```

### *SAMPL Formulation*

Expressed using SAMPL extended syntax, the steps above are simplified. The definition of the random parameter is changed to:

```
random param RProductivity{d in DUTIES, t in TIME}
    dist symmetric(Productivity[d,t] - ProdRange[t],
                   Productivity[d,t] + ProdRange[t]);

option RobustForm Soyster;

AmountWorked{d in DUTIES}:
sum{t in TIME} Productivity[d, t] * Work[d,t] >= WorkRequired[d];
```

The system takes care of generating the artificial variables and the additional constraints automatically, thus allowing the modeller to concentrate on the problem instead of the formal specification of the uncertainty set. To obtain the other for-

`

mulations (Ben Tal and Nemirovsky, or Bertsimas and Sim), the modeller simply uses a different value for the RobustForm option. These two formulations require additional parameters to specify the desired trade-off between optimality and robustness. This parameter is specified in the constraint declaration, as:

```
AmountWorked{d in DUTIES} suffix robustness gamma[d]:
sum{t in TIME} Productivity[d, t] * Work[d,t] >= WorkRequired[d];
```

where gamma is an AMPL parameter containing the chosen robustness value.

## 2.6 Conclusions

In this chapter we have presented a literature review to highlight the data and knowledge flows involved in the definition of a Stochastic Programming problem. We have focused on modelling languages which support the specification of the optimization problem and link with the scenario generation library (see Chapter 4). In section 2.5 we have illustrated (by means of an example) the language features interpreted by the system implemented by the author. The benefit of using these language features over the standard LP model formulation is highlighted.

`

# Chapter 3 Requirements and characteristics of SP solvers

Following the conceptual division between modelling system and solver introduced in Chapter 2, which is widely followed in deterministic optimization too, the implications and the communication requirements of these two systems are hereby discussed. These requirements can be split in three major categories: instance level model representation formats, solution methods and auxiliary data communication. Stochastic Programming models differ from deterministic mathematical programming problems as there are alternative ways to represent them at instance level. The formats and the methodologies to do so are presented in section 3.1 and 3.2 respectively. The structure inherent to SP models is exploited by various algorithms, which are briefly presented in section 3.3, together with some comparative benchmarks. Sections 3.1 to 3.3 are mainly literature review. Section 3.4 shows the model generation capabilities of the implemented system, as an alternative to normal modelling tools. In section 3.5 we present the solver architecture chosen to exploit the availability of such algorithms in an integrated system.

The contribution given by the author is the analysis of possible mappings between model classes and solution methods, in the design of an interface that makes these mappings possible and in its implementation. The other main contribution is the implementation of the model instance generator, which has been used to generate many SP model and its benchmarks, presented in section 3.4.

## 3.1 Instance level formats

Algebraic modelling systems are capable of translating the models from a format which is easily understood and developed by the modeller into a computer readable form which is acceptable by the solvers. In the previous chapter, it has been shown that a model expressed in AMPL (or in any other modelling language) can easily be instantiated over different data sets; a model, at that level, can therefore be considered a template which describes the model's structure. The solvers are programs which implement various algorithms that can be applied only on fully de-

`

scribed numerical models; we call such a fully described model an *instance* of a model. See Figure 13 for a highlight of the role of model instance formats in the context of the comprehensive modelling system diagram reported in Figure 7.

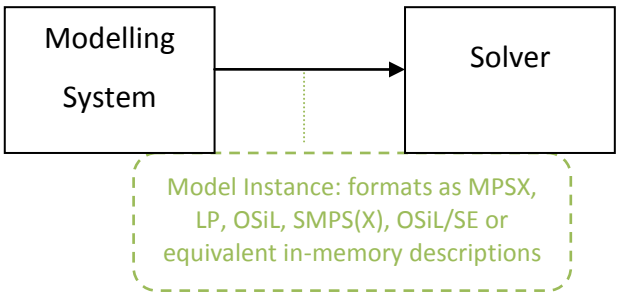



**Figure 13 Instance level formats' role in a modelling system**

### *The MPS(X) Format*

MPS (Mathematical Programming System) and, more precisely, its extended incarnation MPSX (IBM World Trade Corporation, 1976) is an *instance* level format introduced in the 1970s for linear and integer programming problem and it is still today, despite its shortcomings, the de facto standard for LP problems exchange. It played a major role in the development of solution algorithms and software systems for handling these types of problems. The existence of a standard format facilitates the communication of models between modelling and solving software tools, hence enabling the solvers to be seen as interchangeable "black boxes". This in turn allows the designers of solution algorithms to test their software implementations against libraries of benchmark models such as NETLIB (Anon., 2010) and MIPLIB (Bixby et al., 1998).

Stochastic programming models can also be instantiated in MPS format by way of their deterministic equivalent formulations, however there are two major drawbacks to this approach: the deterministic data is replicated for each scenario, causing a unnecessary high data volume and the inherent structure of SP models is scrambled and lost.

`

### *The SMPS format*

The SMPS format (Birge et al., 1987) was introduced in the late 1980s and addresses the drawbacks listed above, following the general structure and guidelines of the MPS standard. The fundamental aim of SMPS is to allow the conversion of existing deterministic linear programs into stochastic linear programs by adding information about the dynamic structure and the randomness, preserving the structure and avoiding data redundancy. The standard specifies the use of three separate files, all organized into fields according to the MPS record layout: the *core* file, the *stoch* file and the *time* file.

The *core* file is a normal MPS(X) file containing the formulation of the problem for a single realization of the random variables, thus a normal LP problem but with a specific ordering of variables and constraints to create a lower block-triangular matrix, which we call the *core* problem. The meaning of *core* and the resulting matrix shape can easily be seen considering the deterministic equivalent formulation of a multistage recourse program, given by the system of equations 1.37 and reported below for ease of reading:

$$\min \sum_{s \in S} \pi_s \sum_{t=1}^{T} c_{ts} x_{ts}$$

$$\begin{aligned}
\textit{subject to}\ \ & A_{11}x_{1s} && = b_1 \\
& A_{21s}x_{1s} + A_{22s}x_{2s} && = b_{2s} \\
& A_{31s}x_{1s} + A_{32s}x_{2s} + A_{33s}x_{3s} && = b_{3s} \\
& \quad . \qquad\qquad . \quad . \quad . && \qquad\qquad \forall\, s \in S \\
& A_{T1s}x_{1s} + A_{T2s}x_{2s} + A_{T3s}x_{3s} + \ \ldots\ + A_{TTs} && = b_{Ts} \\
& \qquad \textit{with} \qquad l_{ts} \le x_{ts} \le u_{ts}\,,\ t = 1,\ldots,T
\end{aligned} \tag{3.1}$$

The data defining this problem can be arranged in an LP formulation for a single realization of the random variables, which we call *core* problem:

$$\min c_1x_1 + c_2x_2 + \ \ldots + c_Tx_T$$

$$\begin{aligned}
\textit{subject to}\ \ & A_{11}x_1 && = b_1 \\
& A_{21}x_1 + A_{22}x_2 && = b_2 \\
& A_{31}x_1 + A_{32}x_2 + A_{33}x_3 && = b_3 \\
& A_{T1}x_1 + A_{T2}x_2 + A_{T3}x_3 + \ \ldots\ + A_{TT} && = b_T \\
& \textit{with} \qquad l_t \le x_t \le u_t\,,\ t = 1,\ldots,T
\end{aligned} \tag{3.2}$$

`

All entries in the matrices $A_{ti}$ and vectors: $c_t$, $l_t$ , $u_t$, $b_t$ can be random, but in practice all but a few entries will be deterministic. This very last fact illustrates the problem of non-necessary data redundancy inherent in the deterministic equivalent formulation, as all the mentioned vectors are entirely specified for each scenario, even though most parts of them are deterministic and therefore remain the same among all scenarios.

The *time* file specifies the dynamic structure of the problem, partitioning the matrix specified in the core file into different stages.

The *stoch* file indicates the distributions of the random parameters, in three possible ways:

**Independent**: The individual coefficients are supposed to be mutually independent, and their distribution is algebraically specified, describing its type and some parameters which depend on the type of the distribution. Some common distributions are natively supported by the standard and, for non-standard distributions, the use of a user defined routine can be specified.

**Blocks**: Considered as mutually independent random vectors.

**Scenarios**: In case of a scenario based recourse problem, the probabilities of the scenarios, their branching stage and the values of the realizations of the random coefficients are listed in this section.

SMPS, in its extended recent version (Gassmann & Schweitzer, 2001), is the most common instance level format for stochastic programming problems. I cannot omit the fact that, even though SMPS is the *de facto* standard instance level language for SP problems, its adoption is not nearly as widespread as the one of the MPS format, slowing down the evolution of specialized solvers for Stochastic Programming. SMPS inherited many limitations that are intrinsic to the aging tabular nature of MPS; to address these problems some alternative formats have been developed in recent years.

`

### *New directions: XML*

The new approach, which takes into consideration contemporary technical assumptions, is based on *eXtended Mark-up Language* (XML). The wealth of functionality and software available for exploiting XML is an important factor in the acceptance of an XML-based mathematical programming standard. XML vocabularies proposed for this role have included OptML (Kristjansson, 2001) and LPFML (Fourer et al., 2005). LPFML has grown into OSiL (Fourer et al., 2006), which can represent linear programs, quadratic programs and general nonlinear program.

OSiL is the instance level format part of the framework O*ptimization Services* (Optimization Services, 2008), which is a general design for XML-based service-oriented, optimization-centred distributed architecture. Optimization Services is intended to be an open source computational infrastructure for running optimization as services on distributed systems and defines some protocols named OSxL (Optimization Services Languages) to standardize three areas of Operational Research namely Optimization Instance Representation, Optimization Communication and Optimization Service registration and discovery. OSiL/SE (Fourer et al., 2009) is an extension to the OSiL schema designed to capture stochastic programming problems with recourse as well as (integrated) chance-constrained models and other forms of optimization under uncertainty. It is a very general design, and it is able to handle continuous as well as discrete distributions, stochastic problem dimensions, various kinds of stochastic processes, linear and nonlinear objectives and constraints and, due to its XML nature, can easily be embedded in SOAP messages which are the native language of web services communications.

### *Example: MPS vs SMPS*

A small example is presented below, to illustrate the difference between the MPS and the SMPS formats (other SP-specific instance level representations as OSiL/SE are different in the grammar used but similar conceptually).

Consider the following two-stage problem instance:

`

$$\min c_1 x_1 + \sum_{s \in S} \pi_s \, c_{3s} x_{3,s}$$

$$\begin{aligned} \textit{subject to } & A_{21}x_1 + A_{22}x_2 && && \leq b_2 \\ & A_{32}x_2 + A_{33,s1}x_{3,s1} && && \geq b_{3,s1} \\ & A_{32}x_2 + && A_{33,s2}x_{3,s2} && \geq b_{3,s2} \\ & A_{32}x_2 + && && A_{33,s3}x_{3,s3} \quad \geq b_{3,s3} \end{aligned} \qquad \textbf{3.3}$$

$$\textit{with } x_1 \geq 0 \;,\; x_2 \geq 0 \;,\; 0 \leq x_{3,s} \leq u_3 \;\; \forall \, s \in S$$

Using the following numeric values:

$S = \{1,..,3\},\ c_1$=1, $\pi_s = \{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\},\ A_{21} = 1, A_{22} = -2, A_{32} = 4,$

$c_{3s} = \{5, 5.5, 6\}, A_{33s} = \{-8, -9, -10\}, u_3 = 200$

The resulting MPS file is:

```
NAME          mps
ROWS
 G  R0001
 G  R0002
 G  R0003
 G  R0004
 N  R0005
COLUMNS
    X1      R0001     1
    X1      R0005     1
    X2      R0001     -2
    X2      R0002     4
    X2      R0003     4
    X2      R0004     4
    X3      R0002     -8
    X3      R0005     -1.666666667
    X4      R0003     -9
    X4      R0005     -1.833333333
    X5      R0004     -10
    X5      R0005     -2
RHS
BOUNDS
 UP BOUND     X3      200
 UP BOUND     X4      200
 UP BOUND     X5      200
ENDATA
```

This file represent, with the correct numerical values, the matrix as in equation 3.3. Five rows have been generated (R0005 is the objective function) and five columns

`

(X3, X4 and X5 correspond to the $x_{3,s}$ in the mathematical formulation, instantiated for all three scenarios). To be noticed, in bold, the replicated values for $A_{32}$ and for $u_3$; replication is one of the drawbacks of this kind of representation, and its amount of it increases with the number of scenarios and the size of the model in general. The information about probability and number of scenarios is lost, the first being now a multiplier for some coefficients in the objective row and the latter driving the repetition of rows.

The SMPS version of the file is more compact, and retains the structure of the problem. The three files which comprise the SMPS standard are listed below.

**COR File: (the "core" structure of the problem)**

```
NAME
ROWS
 N  OBJ
 G  R0001
 G  R0002
COLUMNS
    X1      R0001     1
    X1      OBJ       1
    X2      R0001     -2
    X2      R0002     4
    X3      R0002     -8
    X3      OBJ       -5
RHS
BOUNDS
 UP BOUND     X3      200
ENDATA
```

This file captures the fact that the columns $x_{3,s}$ are logically the same entity, just for different scenarios, as the last three constraints of formulation 3.2. In total, this core instance has three rows and three columns.

**TIM File: (specifies the separation of time periods)**

```
TIME
PERIODS
    X1      R0001                   PERIOD1
    X3      R0002                   PERIOD2
ENDATA
```

The second time period entities start with row R0002 and with column X3.

`

**STO File: (contains the values of stochastic parameters and information about the event tree)**

```
STOCH
SCENARIOS      DISCRETE
 SC S0000001  ROOT       0.3333333333  PERIOD1
 SC S0000002  S0000001   0.3333333333  PERIOD2
    X3      OBJ       -5.5
    X3      R0002     -9
 SC S0000003  S0000001   0.3333333333  PERIOD2
    X3      OBJ       -6
    X3      R0002     -10
ENDATA
```

The file contains the probabilities of each scenario and the values that the stochastic parameters assume in each scenario. Already this simple example shows how an SMPS like format is more suitable to represent SP problems instances, as no repetition is created and, equally importantly, the information about the structure of the problem is kept. The comparison between MPS and SMPS formats is expanded in section 3.4, where the system developed by the author is used to generate instances of some problems in both formats.

## 3.2 Deterministic equivalent

Taking into consideration Section 1.3, which described the deterministic equivalent formulation for various classes of SP problems and Section 3.1, which illustrated how the model instances are passed on to a solver, it is reasonable to draw the conclusion that the formulation, generation and the solution of SP problems could rely solely on the already existing tools for deterministic optimisation.

This approach is called *deterministic equivalent (DEQ*) and it is still the standard way to get a numeric solution of a SP problem. To summarize, the deterministic equivalent formulation is an optimisation problem instance that includes explicit or implicit non-anticipativity for two-stage or multistage recourse problems, and various alternative formulations for (I)CCPs and Robust Optimization problems (see section 1.3). These formulations are agnostic of the structure of the problem and, due to this lack of information, tend to replicate the deterministic data along with the realisations of the random parameters. The resulting model is then passed to a non-specialized solver, which uses an algorithm appropriate to the kind of problem (i.e.

`

simplex or interior point/barrier method) to find a solution. Various drawbacks of this methodology have been mentioned in this thesis, and in this section we highlight a few other issues.

### *Spatial difficulty*

Due to the replication of the data inherent to the deterministic equivalent formulation, its size tends to grow very sharply with the number of scenarios, soon becoming unmanageable. As an example, if the approximation of the distributions are modelled using a fixed number of outcomes at each stage (and the filtration is represented by a tree with an equal number of branches for each node), the number of columns of the problem grows linearly with the number of scenarios, but *exponentially* with the number of stages. Further information on this topic can be found in section 3.4.

### *Computational difficulty*

As the size of the model grows with the increase in the number of scenarios, the deterministic equivalent model becomes increasingly difficult to solve. The inherent block structure of the problem – reported below in 3.4 for a two-stage recourse problem with S scenarios – could be exploited by solution algorithms.

$$A = \begin{bmatrix} A_0 & & & & \\ A_1 & D_1 & & & \\ A_2 & 0 & D_2 & & \\ \vdots & 0 & 0 & \ddots & \\ A_{|S|} & 0 & \dots & 0 & D_{|S|} \end{bmatrix} \qquad \textbf{3.4}$$

There exist efficient implementations of the two major solution methods for *general* LP problems, namely simplex (SSX) and interior-point (IPM) methods. The implementation of both approaches might exploit the structure of the problem in many ways, for example using heuristics to detect the presence of certain structures and taking appropriate actions in case these structures are present. A number of researchers have proposed structure-exploiting solution algorithms; a non-exhaustive list includes (Fourer, 1984), (Birge & Qi, 1988), (Choi & Goldfarb, 1993), (Grigoriadis & Khachiyan, 1996), (Schultz & Meyer, 1991). For some benchmarks of

`

direct methods applied to deterministic equivalent, the reader is referred to Table 11 below.

If a structure exploiting method is applied, the illustrated process of modelling and solving an SP problem seems illogical, because the inherent structure of the model is firstly scrambled by the generation of the deterministic equivalent, and then guessed by the heuristics. As all SP problems with recourse present the same structure, it should be passed from the modelling system to the solver, so that solution algorithms can take advantage of it.

## 3.3 Decomposition techniques

The classes of problems included in the taxonomy shown in Figure 1 have all a peculiar structure and researches have been made to use this structure to speed up the solution process. In general, these methods are called *decomposition* methods, as they don't attempt to solve the whole deterministic equivalent in one go; instead they solve sub problems by applying solution procedures to sub-structures that are inherent in the particular class of problem. Decomposition and structure exploitation is not a unique characteristic of SP problems, there has been considerable research effort diverted to speed up solution of particular models (for example portfolio planning models (Mitra et al., 2007), (Bonami & Lejeune, 2009) and set covering problems (Beasley, 1987), (Beasley & Jornsten, 1992)), and to allow generic structure exploitation (see (Makowski, 2005) (Colombo et al., 2009) for a structure conveying modelling system and (Gondzio & Sarkissian, 2003) (Gondzio & Grothey, 2009) for solution methods based on IPM that can make use of it). The peculiarity of SP problems is that their structure is defined once the class of model (recourse problem, chance constrained problem or integrated change constrained problem) is chosen.

`

### *Two-stage and multi stage recourse problems*

Considering the general formulation of a two-stage recourse problem, given in equations 1.13 and 1.14 and replicated below:

$$
\begin{aligned}
& Z_{HN} = \min cx + E_{\omega}[\, Q(x,\omega)] \\
\text{subject to} \quad & Ax = b \\
& x \geq 0 \\
\text{where} \quad & \\
& Q(x,\omega) = \min f(\omega) y(\omega) \\
\text{subject to} \quad & D(\omega) y(\omega) = d(\omega) + B(\omega) x, \\
& y(\omega) \geq 0, \\
& \omega \in \Omega
\end{aligned}
\qquad \textbf{3.5}
$$

The basic idea behind decomposition algorithms is the approximation of the recourse function $Q(x,\omega)$ which is non-linear, obtained applying cuts to the feasible region of the problem, cuts that are obtained solving many small problems. The L-shaped method (Van Slyke & Wets, 1969) is a version of Benders decomposition (Benders, 1962) adapted to stochastic programming problems; it exploits the fact that the recourse function is convex and polyhedral when it has finite support (i.e. the random parameters follow discrete probability distributions, which is always the case in scenario based recourse problems). For a detailed and in depth descriptions of this method, the reader is referred to the previously mentioned papers, or to (Birge, 1997) which contains a formal descriptions of various solution approaches. Benders' decomposition has also been generalized to multi stage SP problems (Birge, 1985), and it is usually referred to as nested Benders' decomposition.

Various improvements of Benders decomposition methods have been developed, mostly to regularize the "jumps" in the current objective value occurring during the solution process. Regularization techniques can be found in (Rockafellar, 1976), (Lemaréchal, 1978), (Kiwiel, 1985), (Rockafellar, 1976); a more recent development is the *level method* (Lemaréchal et al., 1995), subsequently extended to inexact data (Fábián, 2000) and finally applied to solve SP problems (Fábián & Szőke, 2007).

The software system implemented as practical work for this PhD has been designed in close collaboration with the group implementing FortSP (Zverovich et al., 2009), a

`

solver designed specifically for SP problems. The communication of the models to the solver is achieved through library calls which pass the data in a form that closely resembles the SMPS format. This makes the use of the decomposition techniques implemented in the solver possible. A benchmark of the performance of such solver using Benders' decomposition and level decomposition and of CPLEX 11 used directly to solve the deterministic equivalent is reported in Table 11 below. The table is taken from a study made by our research group (Zverovich et al., 2009) in the development of solution methods for SP problems. It reports, among the solution times for various solution methods, the size of the deterministic equivalent representation for a set of test problems, both in terms of matrix size (rows x columns) and of non-zeroes; some of these problems are taken from standard SP problems collections available on SMPS format (POST (Holmes, 1995), Slptestset and then converted to their deterministic equivalent formulation, some have been generated using SPInE, namely the models SAPHIR (Konig et al., 2007), and WATSON (Consigli & Dempster, 1998). More on the coupling of SPInE and the FortSP follows in section 3.4.

`

| Collection | Name | Scenarios | Size | Non Zeroes | CPLEX (s) | | FortSP (s) | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Simplex | IPM | Benders | Level |
| POSTS | pltexpA2 | 6 | 686x1820 | 3703 | 0.15 | 0.06 | 0.04 | 0.03 |
| | | 16 | 1726x4540 | 9233 | 0.17 | 0.13 | 0.08 | 0.10 |
| | Fxm2 | 6 | 1520x2172 | 12139 | 0.24 | 0.09 | 0.29 | 0.35 |
| | | 16 | 3900x5602 | 31239 | 0.47 | 0.20 | 0.39 | 0.53 |
| | stormG2 | 8 | 4409x10193 | 27424 | 0.32 | 0.38 | 0.60 | 0.83 |
| | | 27 | 14441x34114 | 90903 | 0.87 | 3.33 | 1.93 | 1.65 |
| | | 125 | 66185x157496 | 418321 | 7.00 | 12.33 | 8.38 | 4.99 |
| | | 1000 | 528185x1259121 | 3341696 | 305.81 | 189.53 | 80.20 | 34.46 |
| Slptestset | AITL2 | 25 | 152x204 | 604 | 0.14 | 0.04 | 0.08 | 0.16 |
| | LandS | 3 | 23x40 | 92 | 0.11 | 0.04 | 0.01 | 0.04 |
| | 4node | 16 | 1198x3028 | 7743 | 0.20 | 0.19 | 1.44 | 1.18 |
| | | 32 | 2382x6004 | 15231 | 0.37 | 0.65 | 3.60 | 1.87 |
| | | 64 | 4750x11956 | 30207 | 0.88 | 0.70 | 6.79 | 2.36 |
| | | 128 | 9486x23860 | 60159 | 2.48 | 0.71 | 10.25 | 3.37 |
| | | 256 | 18958x47668 | 120063 | 9.88 | 1.53 | 16.17 | 8.75 |
| | | 512 | 37902x95284 | 239871 | 41.74 | 3.38 | 34.04 | 18.08 |
| | | 1024 | 75790x190516 | 479487 | 457.53 | 7.51 | 69.13 | 36.34 |
| | | 2048 | 151566x380980 | 958719 | 1262.75 | 17.93 | 240.25 | 63.28 |
| | | 4096 | 303118x761908 | 1917183 | 11733.86 | 44.95 | 538.26 | 129.57 |
| | | 8192 | 606222x1523764 | 3834111 | * | 79.73 | 1474.48 | 229.72 |
| | | 16384 | 1212430x3047476 | 7667967 | ‡ | ‡ | 1850.52 | 459.27 |
| | | 32768 | 2424846x6094900 | 15335679 | ‡ | ‡ | 5785.07 | 1029.74 |
| SAPHIR | SAPHIR | 50 | 433932x196253 | 1136753 | 255.03 | ‡ | 465.18 | 369.59 |
| | | 100 | 867832x392453 | 2273403 | 916.04 | ‡ | 701.14 | 533.06 |
| | | 200 | 1735632x784853 | 4546703 | 7579.14 | ‡ | † | 2555.47 |
| | | 500 | 4339032x1962053 | 11366603 | ‡ | ‡ | 2556.06 | 2339.76 |
| | | 1000 | 8678032x3924053 | 22733103 | ‡ | ‡ | 4294.47 | 4650.19 |
| WATSON | WATSON | 128 | 41483x75151 | 188828 | 1.44 | 1.71 | 0.92 | 0.92 |
| | | 256 | 82955x150287 | 377628 | 3.66 | 3.89 | 1.58 | 1.59 |
| | | 1024 | 331787x601103 | 1510428 | 14.44 | 20.27 | 5.12 | 5.31 |

**Table 11 Performance of various algorithms**

* Failed to solve due to timeout † Failed to solve due to numerical difficulties

‡ Failed to solve due to insufficient memory

`

### *(Integrated) Chance Constrained Problems*

Chance Constrained Problems have a deterministic equivalent formulation (see section 1.3), which follows immediately from the general formulation. The formulation requires the introduction of binary variables for each scenario to indicate the occurrences of violations in the constraints and the creation of an extra constraint to count and relate them to probability. These stratagems are mechanical, once the description of the chance constraint has been given, and the formulation highly redundant; a solver could thus generate all these structures internally, without the need of over sizing the communicated model.

An equivalent formulation for ICCPs is available (see section 1.3) however for this class of problems cutting plane approaches are available (Haneveld & van der Vlerk, 2006). The proposed cutting plane algorithm speeds up the solution process and, as for recourse problems, offer a greater scalability than direct methods.

## 3.4 Model generation

The implemented system ran through a long series of QA models, to prove the correctness of the generated model instances and to benchmark how the generation ability scales up with the number of scenarios. It is easily seen that the model size increases rapidly with the number of scenarios; in fact the dependence is linear, with a coefficient that depends on the relative size of the first and second stage matrix.

All the models have been written in AMPL as deterministic equivalents and in SAMPL; the two representations have been instantiated using respectively AMPL and SPInE. Finally, MPS or SMPS files have been generated as appropriate. The following table shows some statistics, obtained using an Intel Core 2 Duo 2.8 Ghz, 4 GB RAM, Windows 7 64 bits.

`

| Model | Core Size | | | Deterministic Eq. Size | | SMPS – SPInE | | MPS -AMPL | |
|---|---|---|---|---|---|---|---|---|---|
| | Cols | Rows | Scen | Columns | Rows | Size (Bytes) | Memory Diff. [kB] | Size (Bytes) | Memory Diff. [kB] |
| Dakota | 9 | 10 | 3 | 27 | 48 | 1620 | 1280 | 4129 | 704 |
| ICCPDakota | 9 | 13 | 3 | 39 | 40 | 2104 | 1280 | 4313 | 1216 |
| ICCPDakota10 | 9 | 13 | 10 | 123 | 124 | 3660 | 1280 | 13428 | 912 |
| Power | 12 | 9 | 4 | 39 | 27 | 1742 | 1280 | 4024 | 1024 |
| ALMsmall | 14 | 8 | 2 | 28 | 30 | 1482 | 1280 | 2490 | 92364 |
| ALMTwoStage | 280 | 103 | 360 | 100800 | 62280 | 2667463 | 60736 | 9075248 | 111860 |
| Prod2Stage | 156 | 108 | 8 | 1248 | 1176 | 20938 | 1280 | 147639 | 848 |
| prodMStage | 156 | 108 | 8 | 1248 | 1440 | 21034 | 1280 | 157968 | 0 |
| prodCCP | 120 | 108 | 8 | 1248 | 1212 | 20134 | 2560 | 158524 | 720 |
| prodICCP | 120 | 108 | 8 | 1536 | 1212 | 20244 | 1280 | 157199 | 784 |
| Informer | 3 | 2 | 10 | 42 | 11 | 1350 | 1280 | 2045 | 1024 |
| saphir2 | 3971 | 8704 | 2 | 3974 | 8704 | 872168 | 2048 | 831538 | 5440 |
| saphir10 | 3971 | 8704 | 10 | 39236 | 86788 | 1136119 | 89984 | 8347386 | 22384 |
| saphir20 | 3971 | 8704 | 20 | 78416 | 173548 | 1468465 | 136768 | 16773657 | 49860 |
| saphir50 | 3971 | 8704 | 50 | 195956 | 433828 | 2460251 | 153024 | 42091503 | 122232 |
| saphir100 | 3971 | 8704 | 100 | 391856 | 867628 | 4118397 | 185728 | 84287526 | 241548 |
| saphir200 | 3971 | 8704 | 200 | 783656 | 1735228 | 7580814 | 185728 | 1.7E+08 | 461324 |
| saphir500 | 3971 | 8704 | 500 | 1959056 | 4338028 | 17813339 | 229056 | 4.26E+08 | 1188332 |
| saphir1000 | 3971 | 8704 | 1000 | NA | NA | 36059441 | 320128 | NA | NA |
| saphir2000 | 3971 | 8704 | 2000 | NA | NA | 72195561 | 571008 | NA | NA |
| saphir5000 | 3971 | 8704 | 5000 | NA | NA | 1.8E+08 | 870231.3 | NA | NA |

**Figure 14 Model generation statistics**

The shaded columns show data from the deterministic equivalent formulation, generated using AMPL while the white columns contain statistics on the SAMPL version of the same model, generated using SPInE. It is worth noticing that the last three models couldn't be generated using AMPL 32 bits, as the requested memory hit 2GB, limit for 32 bit processes in windows.

The column **Size** shows the size in bytes of the generated MPS or SMPS model, the column **Memory diff.** shows the memory allocated above the normal runtime environment of AMPL or SPInE in kB. I have chosen to show the peak memory variation which occurs while generating the model over the "empty" runtime environment instead of the peak memory allocation because the two applications are coded using completely different systems (C in case of AMPL, C# under .NET framework 3.5

`

in case of SPInE). For completeness, the initialization of AMPL takes 59 MB, while for SPInE the figure runs up to 166 MB.

It is clear however that, as the number of scenario increases, the generation of the model using SPInE and its representation using SMPS become increasingly advantageous over the deterministic counterparts. The graph in Figure 15 below shows the memory usage differential for the model Saphir, as the number of scenario increases.

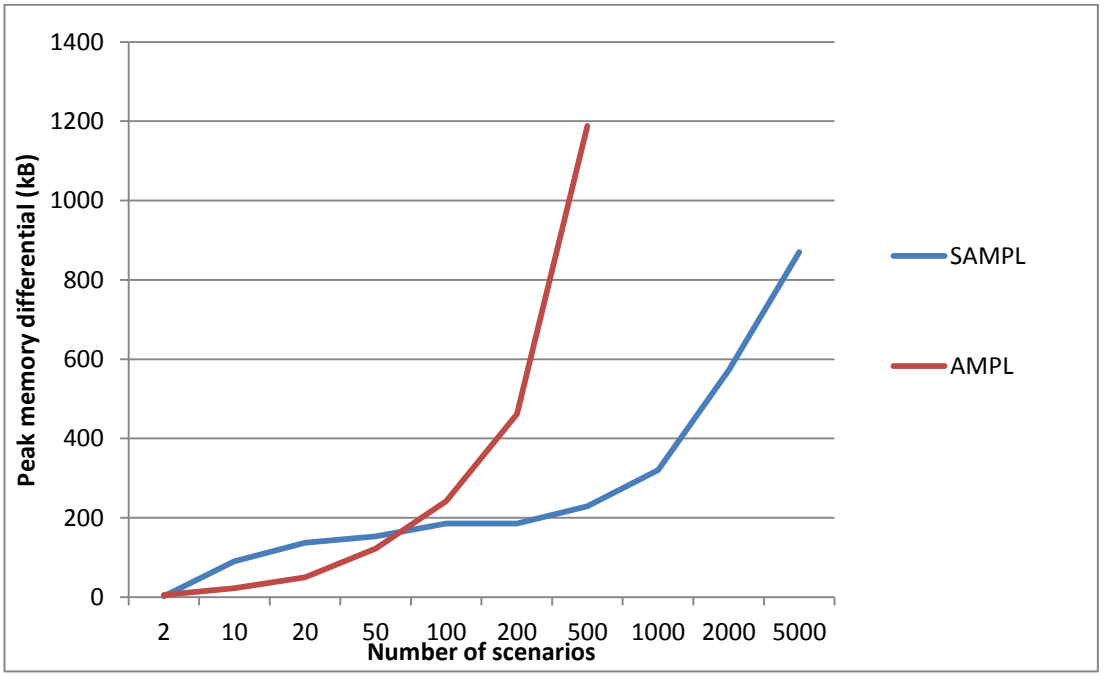



**Figure 15 Comparison of memory usage in model generation**

It can be seen that generating the deterministic equivalent version is advantageous at the beginning of the graph, due to the efficiency of AMPL and its runtime environment but, as the number of scenarios increases, the generation of the SAMPL model using SPInE becomes clearly more efficient.

## 3.5 Solver architecture and interface

The previous sections of this chapter explain that there are well defined classes of model that could benefit by the application of specific solution approaches. An effective software tool for stochastic programming therefore identifies the class of model which is being implemented and takes the appropriate steps to ensure its fast and efficient solution.

`

FortSP, the solver mentioned in various section on this thesis, has been developed in parallel with the development of SPInE, by the SP team in CARISMA. The latest additions and their recent redesign followed the same pattern, resulting in a great interoperability of the two systems. The software system FortSP (Ellison et al., 2009; Zverovich et al., 2009) offers a choice of algorithms, which have different performance and that can be used on different classes of models:

- Benders decomposition (L shaped method)
- Variant of level decomposition
- Nested Benders decomposition
- Cutting plane algorithm for ICCPs
- Deterministic Equivalents generation and solution

The first two are applicable to two-stage SP problems with recourse, the third one to multi stage SP problems, the fourth one to ICCPs and the fifth to all the classes of models. All these methods share the need of an external solver to solve the sub problems or the whole problem (in case of deterministic equivalent). The classes of these problems (Linear Programs, Quadratic Programs, Mixed Integer Programs, Quadratic Mixed Integer Programs, and Second Order Cone Programs) depend on the considered class of starting (SP) model and on the solution method chosen; FortSP has a plug-in system to connect to various external solvers, each of them can have different capabilities in terms of solvable models. The process of chosen the solution method could be at least partially automated, as the requirements in terms of solvers will be known by the modelling system, if the meta-information regarding the model is retained.

A mapping between various methods is possible, and it is reported in Figure 16 for the methods currently implemented in FortSP. Retaining the information about the starting class of the model is possible in a modelling system using specialized syntactic constructs like the ones available in SAMPL (see section 2.5). Aware of this information, the implemented system SPInE can easily exclude some combinations of solution methods and sub solvers. The mapping is currently stored in an external XML file; as new solution methods and solvers are made available, it can be extend-

`

ed to include the new possibilities. Another, more elegant, approach would be to define a simple service discovery API, which would enable solution methods and sub solvers to be self-descriptive. In this way, the update of the mapping could be automated; the system has not been developed in this way because, due to the limited amount of entities and rules, it is easy enough to manually update the XML file.

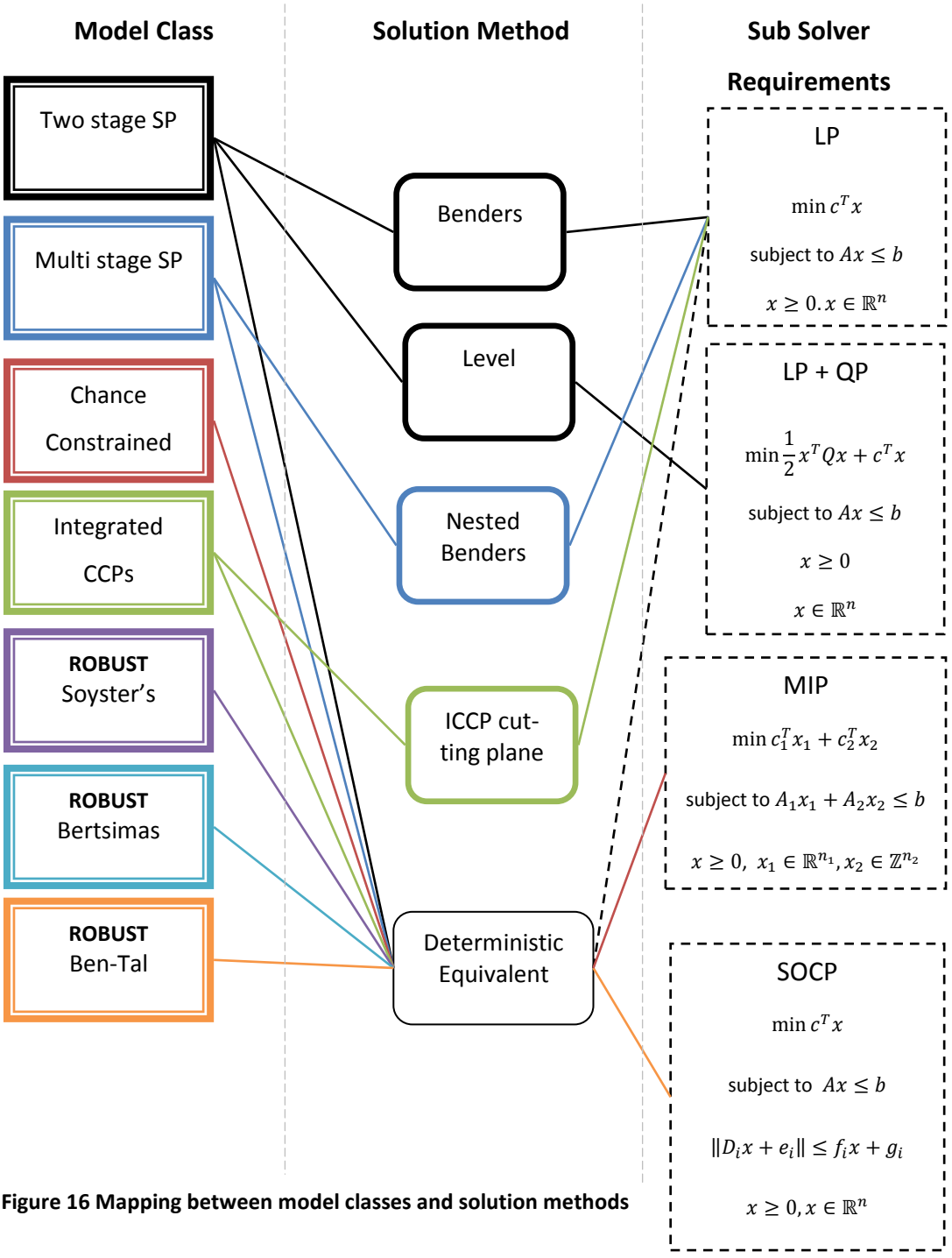



**Figure 16 Mapping between model classes and solution methods**

`

The interface that SPInE uses to communicate with FortSP is shown, as displayed by Visual Studio, in Figure 17 below.

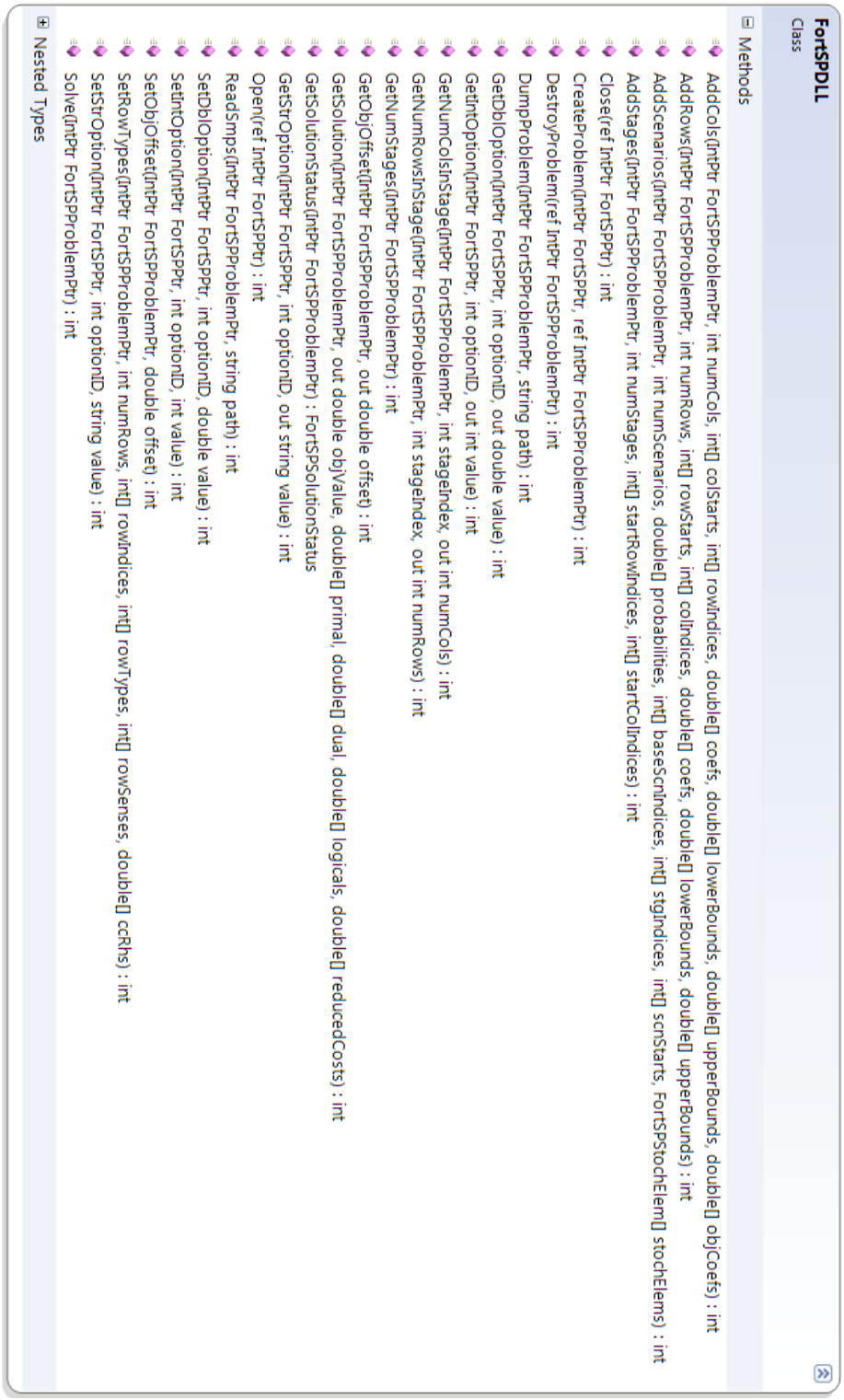



**Figure 17 Interface between SPInE and FortSP**

`

Once generated, the procedure to pass the model to the solver is quite simple. SPInE has the **`Model`** class in memory, which contains all the needed information about the model instance. An instance of the solver is created, then the core problem is passed row-wise, in a sparse row representation. The stochastic information (the parameters which distinguish the scenarios) is then passed in a separate data structure, and finally the objective sense and the eventual type of chance constraints or integrated chance constraints are passed with separate library calls.

The code snippet below, taken from the actual SPInE C# codebase shows this part of the program. It can be noted that, although the interface has been designed closely interacting with the developers of FortSP, there are some differences between the data structures of the solver and of SPInE. This can be noticed in the code at the moment of passing the stochastic elements: a local function (**`createStochElementsArray`**) is called to convert the stochastic elements from SPInE's internal representation to FortSP's.

```
// Create an instance of the solver
result = FortSPDLL.FortSP_Open(ref FortSP);

// Create a pointer to the solver rep of the problem
result = FortSPDLL.FortSP_CreateProblem(FortSP, ref FortSPProblem);

// Add the columns with their lower bounds, upper bounds and costs,
but no A matrix
result = FortSPDLL.FortSP_AddCols(FortSPProblem, model.nc, null,
null, null, model.lob, model.upb, model.cost);

// Get the sparse row representation from the SPInE model class
int [] rowstarts;
int [] colins;
double []values;
model.getSparseRowRepresentation(out rowstarts, out colins, out
values);

// Pass the obtained A matrixs to the solver
result = FortSPDLL.FortSP_AddRows(FortSPProblem, model.nr,
rowstarts, colins, values, model.lhs, model.rhs);

// Add the stages information
result = FortSPDLL.FortSP_AddStages(FortSPProblem, model.nt - 1,
      model.rowstart, model.colstart[];

// Create the array of stochastic elements in the format FortSP
expects it
```

`

```
FortSPDLL.FortSPStochElem[] stochElements =
createStochElementsArray(model);
int[] newstoinds = new int[model.ns + 1];
model.stoind.CopyTo(newstoinds, 0);
newstoinds[model.ns] = model.nd;

// Pass the stochastic elements to the solver
result = FortSPDLL.FortSP_AddScenarios(FortSPProblem, model.ns,
model.prob, model.basescen, model.stage, model.stoind,
stochElements);

// Set objective sense
FortSPDLL.FortSP_SetObjSense(FortSPProblem, (int)
model.objectiveSense);

// If present, set the types of chance constraints or integrated
change constraints
if (model.CCnumberOfCCs > 0)
{
result=FortSPDLL.FortSP_SetRowTypes(FortSPProblem,
model.CCnumberOfCCs , model.CCrowindices, model.CCrowTypes,
model.CCrowDirection, model.CCrhsValue);
}
```

This kind of interface is easy yet functional enough, and it has proven to be solid across many months of testing.

## 3.6 Conclusions

In this chapter we have highlighted the differences between deterministic and SP problems in terms of model instance generation, representation and solution. We have shown that the developed system has the means to automatically generate efficient SP model instances, and the benefits of such technique are shown in the benchmarks set out in section 3.4. Solution speed and memory usage can greatly benefit by the use of decomposition methods, and the interface devised to automatically select and use a method depending on the problem at hand is shown in section 3.5.

`

`

# Chapter 4 Requirements for a Scenario Generation library

The computational solution of SP problems, except for some trivial cases, cannot be obtained with continuous distributions of the random model parameters. Therefore in almost all cases discrete distributions are used, leading to scenario based stochastic programming problems, and indeed the discrete scenario based models are supported by most solution methods. Moreover, the number of scenarios of the discrete distributions is limited by the available computing power, and therefore these distributions have to be approximated by a limited number of outcomes. The approximation is usually in the form of a *scenario tree,* and the mathematical/software tools used to generate them are referred to as *scenario generators*. In this chapter we present the requirement analysis for a scenario generator library. Our aim in defining a scenario generator library and capturing alternative SG methods are as follows: through the many procedures listed in the library, we capture well established knowledge of leading methods, and we connect them to a decision model in a user friendly way; this system has been implemented in SPInE. The chapter starts with a formalization of scenario generators, given in section 4.1. Two sections of literature review then cover respectively the classification of scenario generation methods (section 4.2) and their desirable properties (section 4.3). Finally the scenario generation library is presented and described in two sections 4.4 and 0. These sections respectively set out successive formalizations of the concept of scenario generator, define the software requirements and present some excerpts of the implementation of the library itself. Section 4.6 gives then a formal introduction to the new syntax introduced in SAMPL to integrate scenario generation into the decision model.

The author's contributions in respect to this chapter are the analysis, design and implementation of the scenario generation library, as reported in sections 4.4 and 0 and the specification of the language extensions to support it at a modelling level (4.6).

`

## 4.1 Scenario Generators: a modelling perspective

The creation of the scenario tree used in scenario based stochastic programming is usually performed by specialized applications called *scenario generators*. The process of capturing the computational SP models involves therefore two distinct modelling steps, which are:

(i) Expressing the logic of the application's domain into a (SP) decision model
(ii) Representing the randomness properties of the application's domain with stochastic processes and create scenario trees

Depending on the domain of application, the domain experts typically apply well established models to specify the random model parameters, finely tuned and matching the problem at hand. For instance in financial applications CAPM, GARCH, Geometric Brownian Motion, Regime Switching Markov models have been extensively used. For a more comprehensive description of scenario generators for financial models, the readers are referred to some recent research reports and publications of CARISMA (Roman et al., 2009), (Mitra, 2009). Other domains have other typical models; a short list of such domains and common techniques is given in Table 8.

The next necessary step is to connect the chosen scenario generator to the decision model; this is usually performed by importing the generated scenario tree in the form of data structured typically as a multidimensional table into the modelling system. This procedure is often time consuming and error prone; creating an integrated software tool for SP should avoid this manual step, thus a further analysis of the connection between SGs and decision model is required. Figure 18 below illustrates the two conditions that are vital for the implementation of such a connection: a structural compliance between the SGs parameters and the algebraic model and the communication of controls between modelling system and scenario generator. The first condition (Compliance) is related to modelling, and is expanded in the rest of this section, while the second (Control) is related to the software infrastructure, and is discussed in sections 4.4 and 4.5.

`

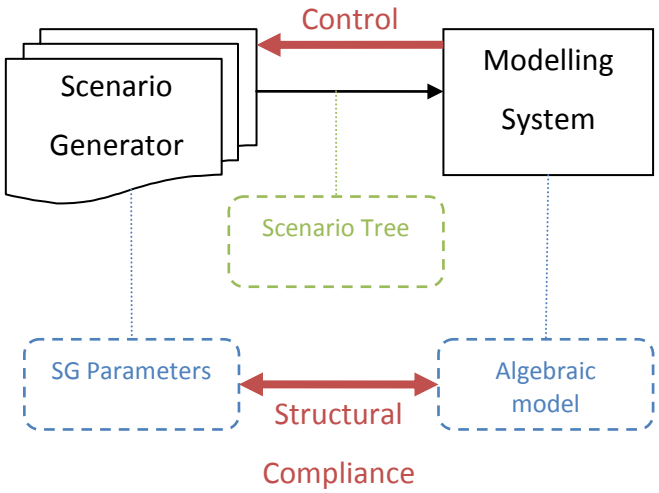



**Figure 18 Link between Scenario Generators and Modelling System**

A scenario generator *f* captures in a procedural form a domain-specific model of randomness. In particular it very often uses (i) historical information, (ii) an event tree structure and (iii) some other specific parameters needed by the model of randomness. We therefore separate the three groups of parameters as:

(i) *H* History

(ii) $\tau$ Event Tree shape

(iii) $\pi$ Remaining Parameters, needed by the model of randomness

Using this notation, the set of scenarios $\Xi$ can be seen as the output of the generation procedure, as:

$$\Xi = f(H, \tau, \pi) \qquad \textbf{4.1}$$

The event tree structure is specified in the decision model too; it is actually enforced through non-anticipativity constraints or by appropriate variable definitions in normal modelling languages, or defined as a stand-alone entity in languages with specific constructs for SP (see section 2.5 for examples). We define the tree structure expressed to the decision model as $\tau'$.

The two trees need to be congruent; in other words, the tree structure $\tau$ assumed by the scenario generator has to be the same as the tree structure $\tau'$ specified in the optimization model (see Figure 19).

`

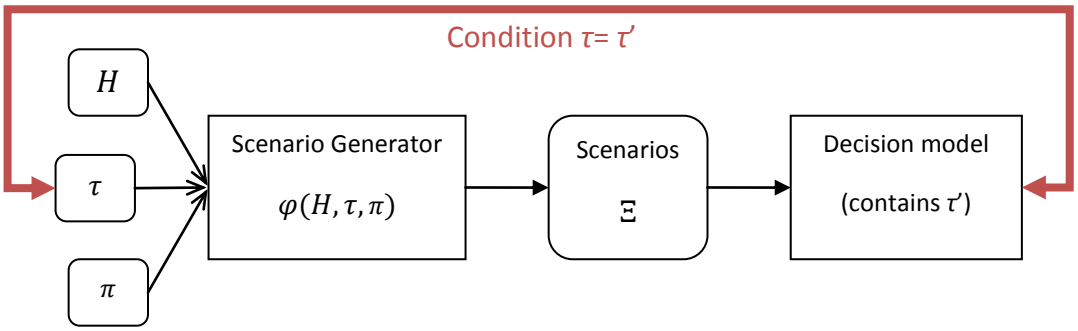



**Figure 19 Structural compliance between model and SG**

Logic dictates that there are two ways to ensure this consistency: the first is to check whether the two definitions of $\tau$ and $\tau'$ coincide; the second is to define the tree structure just once, thus eliminating any possibility of inconsistencies. In the current version of SPInE, the tree structure is defined using the SAMPL construct `tree` in the decision model and it is automatically communicated to the scenario generators used. This might not be the best possible choice, as scenario generators with integrated scenario reduction techniques or with internal sampling methods (see the next section), might be able to provide a tree structure automatically given a desired level of precision in the representation of the underlying stochastic process. The approach of getting $\tau$ from the scenario generators is currently under research.

## 4.2 Overview of SG methods

Scenario generation is possible following various general approaches; on top of these techniques there are methods to then reduce the number of scenarios to a tractable case. The common aim of all methods and techniques is to be able to approximate a distribution with a treatable scenario tree; the following categorization is largely taken from (Kaut & Wallace, 2003).

1. **Generating scenarios**
    - **Conditional sampling**

      At every node of the scenario tree, realizations of the stochastic process $\{\tilde{\xi}_t\}$ are sampled, either by sampling directly from the distribution of $\{\tilde{\xi}_t\}$ or by evolving the process discretely, according to a formula of the type $\tilde{\xi}_{t+1} = z(\tilde{\xi}_t, \varepsilon)$ where $\varepsilon$ is the current random vec-

`

tor. Traditional sampling methods have a limitation: when sampling from more than a random variable, there is the need to sample every marginal separately and combine them afterwards, generating a tree whose size grows exponentially with the dimensions of the random vector.

- **Sampling from specified marginals and correlations**

  To overcome the difficulties in generating multivariate vectors, especially if correlated, these methods require the specification of the marginal distributions and the correlation matrix. Copulas are often used in these methods to bind together the various marginals.

- **Moment matching**

  If the distributions are not known, they can be described using their moments (mean, variance, skewness etc.), the correlation matrix in case of multivariate vectors and possibly other statistical properties. A discrete distribution can then be constructed that matches the given statistical properties (Hochreiter & Pflug, 2007), (Smith, 1993). Bootstrapping can be seen as a moment matching technique, in which the desired distribution is created by mean of values sampled directly from the original distribution's values.

- **Path-based methods**

  In these methods, whole paths (or fans) are generated evolving the stochastic process over time, one for each scenario. These scenarios have then to be clustered into a scenario tree of the desired shape.

**2. Related techniques**

- **Clustering**

  Clustering is the technique used to convert a set of scenarios in form of fans to a scenario tree. See (Dupačová et al., 2000) or (Heitsch & Römisch, 2009) for a combined clustering/reduction approach.

`

- **Internal sampling methods**

  These methods differ from the others, as the sampling of scenarios is performed during the solution procedure. Most important methods of this kind are stochastic decomposition (Higle et al., 2009), importance sampling in Benders decomposition and stochastic quasi-gradient methods.

- **Scenario reduction**

  Scenario reduction is a method to decrease the size of an already generated scenario tree, trying to find a scenario subset of prescribed size that is closest to the initial distribution in terms of defined probability metrics (Dupačová et al., 2003), (Henrion et al., 2008), (Heitsch & Römisch, 2009).

In general, all these methods can be divided into Statistical models or other models, depending whether the underlying random process is describing the real world based on scientific/mathematical theories of the given field, or just based on statistical properties (i.e. moments) of observed historical data.

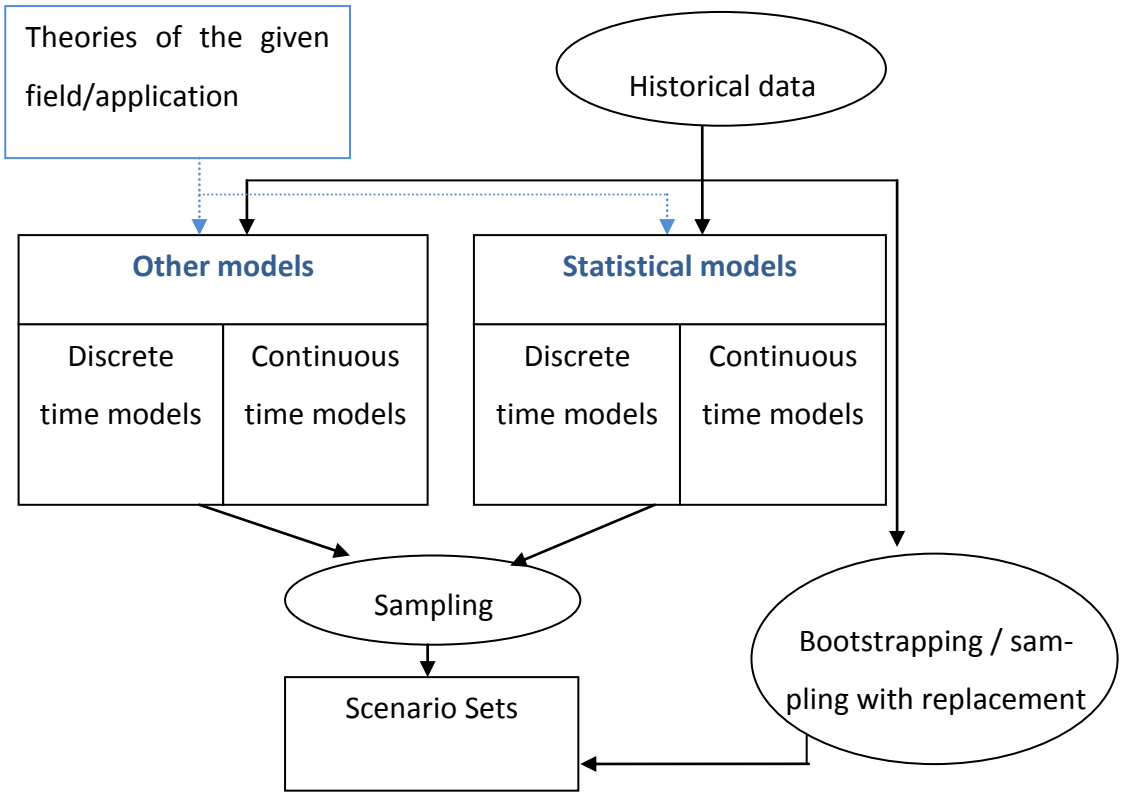



**Figure 20 Scenario generation methods**

`

There are some common models of randomness, whose use spreads across various application fields and are therefore listed below. They are not direct scenario generation techniques, as they generate time series, but continuous time models from which a scenario tree can be derived using sampling or clustering methods.

**Diffusion processes**

Diffusion processes are widely used in finance to model the future evolution of stock prices, interest rates and mortality ratio. They are continuous time models, and include:

*Wiener Processes (Brownian motion)*

Brownian motion is one of the simplest stochastic processes, and it is described in a mathematically convenient form. It is traditionally regarded as discovered by Robert Brown in 1828 (Brown, 1828) applied to the movement of pollen particles, mathematically formalized in 1880 by Thorvald N. Theile (Thiele, 1880), but its first well known application is due to Albert Einstein in 1905 as a description of the movement of small particles in a stationary fluid (Einstein, 1906).

A Wiener process is a stochastic process which is defined as:

$$dz = \varepsilon\sqrt{dt}, \qquad where\ \varepsilon \sim N(0,1) \tag{4.2}$$

and *dz* is the drift of the value of the process in dt. The process has the following important properties:

1. It is a Markov process, so future probability distributions depend only on the current value of the process and not on past values or other information
2. The increments over two defined time intervals are independent
3. Changes in the process over any finite time interval are normally distributed with a variance which increases linearly over time

`

*Generalized Wiener Processes (Brownian motion with drift)*

A generalized Wiener process is defined as follows:

$$dx = \alpha dt + \sigma dz \qquad \textbf{4.3}$$

where *dz* is the increment of a standard Wiener process, $\alpha$ is the drift parameter and $\sigma$ is the standard deviation. Over any time interval Δt, the corresponding Δx is normally distributed with mean $\alpha dt$ and variance $\sigma^2 dz$.

*Ito Processes*

The Ito process is a generalization of Brownian motion with drift and can be expressed as:

$$dx = a(x,t)dt + b(x,t)dz \qquad \textbf{4.4}$$

where *dz* is the increment of a standard Wiener process, $a(x,t)$ and $b(x,t)$ are the drift and the standard deviation expressed as functions of the current state and time. A particular case of the Ito process is the Geometric Brownian Motion (GBM), where $a(x,t) = \alpha$ and $b(x,t) = \sigma$.

The use or one or the other models depends on the assumptions made on the described reality: in case of modelling share prices, if we assume that the expected percentage return and the variance of the return are independent from the current price, the price can be modelled by a GBM, otherwise not.

**Time series**

Time series are commonly used to estimate parameters which explain the behaviour of a random variable based on past observations. Three broad category of models of practical importance are the autoregressive (AR) models, the integrated models (I) and the moving average (MA) models, which can be combined and which all assume linear dependence between the current data point and the previous one(s). Non-linear dependency is possible, like in the conditional heteroskedasticity models, in which the variance varies over time.

`

*Autoregressive models: AR(p)*

AR model of order *p* assume that the current value of a random variable depends solely on the past *p* observations of the same variable. The general form of an *AR(p)* process is:

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t\,, \qquad where\ \varepsilon_t \sim IIDN(0, \sigma^2) \tag{4.5}$$

where $X_t$ is the value of the process at time *t*, $\varphi_i$ are the parameters of the model, *c* is a constant and $\varepsilon_t$ is white noise. The *AR(1)* model is known as *random walk.*

*Moving Average models: MA(q)*

The moving average model is conceptually a linear regression of the current value of the random variable against previous (unobserved) white noise terms. A moving average model of order *q* is expressed as:

$$X_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}\,, \qquad where\ \varepsilon_t \sim IIDN(0, \sigma^2) \tag{4.6}$$

where $X_t$ is the value of the series at time *t*, $\theta_i$ are the parameters of the model, $\mu$ is the mean of the series and $\varepsilon_t$ are white noise error terms.

*Autoregressive Moving Average models: ARMA(p,q)*

Autoregressive moving average models are defined as a combination of AR and MA models. An ARMA(p,q) model is therefore expressed as:

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}, \qquad where\ \varepsilon_t \sim IIDN(0, \sigma^2) \tag{4.7}$$

where all the symbols have the meaning defined for the AR and the MA formulations.

*Autoregressive Conditional Heteroskedasticity models: ARCH(q)*

`

This class of AR models assume that the random variable is characterized by non-constant variance over time, that is: the variance of the current error is considered to be a function of the values of the previous time periods' errors. The process is then modelled with an AR(q) model, as:

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t \qquad \textbf{4.8}$$

where $\varepsilon_t = \sigma_t z_t$, with $z_t \sim IIDN(0,1)$, and with:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 \qquad \textbf{4.9}$$

where $\alpha_0 > 0$ and $\alpha_i \geq 0, i > 0$.

*Generalized Autoregressive Conditional Heteroskedasticity models: GARCH(p,q)*

A GARCH(p,q) model assumes an ARMA(p,q) model for the error variance, therefore the error term for the model process is given by:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{i=1}^{p} \beta_i \sigma_{t-i}^2, \qquad where\ \varepsilon_t \sim IIDN(0,\sigma^2) \qquad \textbf{4.10}$$

## 4.3 Desirable properties

The quality of the decision resulting from the stochastic programming problem is, as can be inferred from section 4.1, dependent on both the decision model and the scenario generation process. Some desirable properties for scenario generators are described in (Kaut & Wallace, 2003) and (Zenios, 2006); these can be summarized as:

**1. Correctness:** The generated scenario sets should be correct representations of our random parameters' distributions; not knowing the distribution leads us to different descriptive models which give alternative representations of our parameters' dynamics. It is important to choose the model that best captures the aspects of the dynamics of the random parameters that are important in the context of the decision problem.

`

**2. Consistency:** In case of multiple related random parameters, the values of these, under any particular scenario, should be consistent with each other. This issue arises when there are domain specific rules which apply to two or more of the generated random parameters: the generated scenarios, which include values for both the parameters, should be consistent with the domain rules (i.e. in finance, generated prices for different type of assets might have to satisfy the *arbitrage free* condition, or other logical inconsistencies between parameters' values).

**3. Stability:** Stability for a scenario generation method is considered in respect of a particular decision model. A scenario generator is stable in respect to a decision problem if the decisions which are outcome of the decision problem do not vary significantly across multiple runs. Defining the decision model as a simple single stage model as:

$$\min_{x \in X} F(x, \tilde{\xi}_t) \qquad \textbf{4.11}$$

where $\tilde{\xi}$ is a stochastic variable, and using a notation where $\{\tilde{\xi}_t\}$ is a stochastic process, $\breve{\xi}$ is a discrete stochastic variable and $\{\breve{\xi}_t\}$ a discrete stochastic process (thus a scenario tree), Kaut and Wallace (2003) define, over K generated scenario trees $\breve{\xi}_{tk}$ and the same number of obtained solutions of the problem $x_k^*$, k=1…K

*in-sample* stability:

$$F(x_k^*, \breve{\xi}_{tk}) \cong F(x_l^*, \breve{\xi}_{tl}) \qquad k, l \in \{1, \dots, K\} \qquad \textbf{4.12}$$

and *out-of-sample* stability*:*

$$F(x_k^*, \breve{\xi}_{tk}) \cong F(x_l^*, \breve{\xi}_{tl}) \qquad k, l \in \{1, \dots, K\} \qquad \textbf{4.13}$$

The in-sample stability tells us therefore how stable the scenario generator is when used with the considered decision model or, in other words, how much the objective function value changes when solving the decision model using different trees generated with the same scenario generator and the same parameters.

The out-of-sample stability evaluates the solutions obtained through the scenario generator against the *real* distribution; in the (very realistic) case that the real dis-

`

tribution is not known, the solutions can be evaluated against another scenario generator that has proven to be reliable with the current decision model or back-tested against historical data.

## 4.4 An abstract view of SGs

Starting from the basic points given in section 4.1, a second step of abstraction seems necessary to achieve a compact yet extensible architecture to seamlessly connect scenario generators to a modelling system. The interested reader would probably have noticed that just one of the connections in Figure 19 has been described in that section. The second one needs a more thorough formalization of the abstraction already presented to represent scenario generators. It has to be noted that the structural compliance condition is enough to guarantee that a decision model can work together with a predefined scenario generator; the second connection is mandatory when the functionalities of a *scenario generator library* are introduced. A scenario generator library is an extensible collection of scenario generators among which the modeller can choose for generating the random data.

The problem of creating an abstraction and some functionality requirements to enclose a scenario generation library and its linking to a decision model is hereby discussed. The new formalism is introduced through subsequent extensions in the multiplicities of the entities involved:

| Number of scenario generators | Number of random parameters |
|---|---|
| **Single scenario generator** | Single random parameter |
| **Multiple scenario generators** | Single random parameter |
| **Multiple scenario generators** | Multiple random parameters |

**Table 12 SG abstraction steps**

Please note that in this section, the mentioned models of randomness (AR, ARMA) are not intended to completely characterize the scenario generator, as these models generate time series data. This output will then have to be sampled to obtain the scenario tree (see Figure 20 and its comment). The word “based”, as in “AR(2) based scenario generator” serve to further stress this fundamental distinction.

`

### *Single scenario generator – Single random parameter*

From a functional perspective, a scenario generator (if not internal sampling based) is an independent module that has a few inputs and outputs a data structure containing random parameter values in the form of a scenario tree, as in Figure 21.

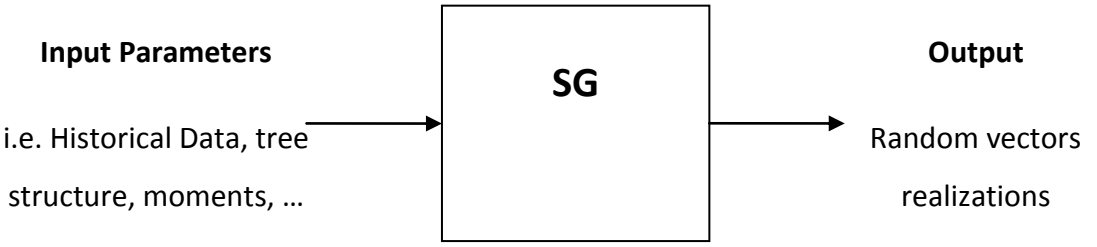



**Figure 21 Scenario generator functional perspective**

To characterize a scenario generator, we therefore should proceed by identify inputs, outputs and the actions it can perform. Examining a first example, a simple AR(2) based scenario generator, the needed parameters (see equation 4.5) are: $\tau, \varphi_1, \varphi_2, c$ and $X_0$ where $\tau$ describes the scenario tree shape. If the above scenario generator is able to estimate its parameters from historical data, then the input is simply $\tau, H$ (denoting the historical data). For an ARMA(1,1), the needed parameters would be $\tau$, $\varphi_1, \theta_1, c$ and $X_0$ or again simply $\tau, H$. For all these models, the output is the scenario tree ξ. A model able to generate multivariate vectors, will need the dimension $\kappa$ of the vector to be generated as well as the parameters specific to the stochastic process used, that we summarize with $\pi$. These can be schematized as in Figure 22 below.

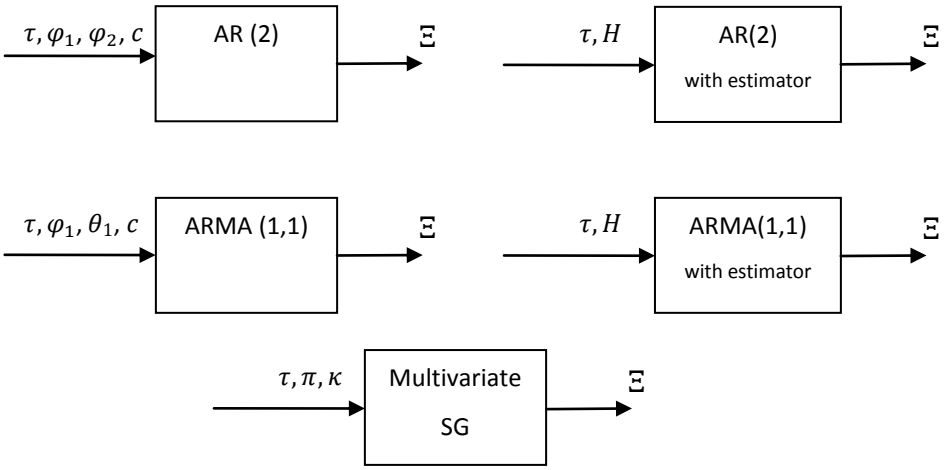



**Figure 22 Scenario generators as black boxes**

`

The procedure of using a scenario generator can be expressed as:

$$\Xi = f(H, \tau, \pi) \qquad \textbf{4.14}$$

where $\tau$ is the tree structure shape, H is the historical data, $\pi$ is a set of parameters specific to the scenario generator, $\Xi$ is the set of generated scenarios. From now on, we will consider among the set of possible parameters $\pi$, the recurring ones as the historical data *H* and the dimension of the random vector *κ*, which are often present in scenario generation procedures and, most importantly, that could be specified as part of the data of the model. The procedure can then be written as $f(\tau, \pi)$.

Introducing notation that is used in the next section, we can therefore fully define a scenario generator using:

- $f$ identifies a scenario generator procedure
- $\pi \stackrel{\text{def}}{=} \{\pi_1, \dots, \pi_n\}$ identifies the set of n parameters needed by $f$
- $\Xi$ is the scenario set generated by $f$

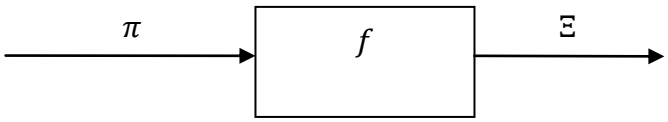



**Figure 23 Abstract representation of a scenario generator**

### *Multiple scenario generators – Single random parameter*

This first extension extends the structure to support multiple scenario generators, that we call a *scenario generators library.* They could all possibly need different inputs and, whatever scenario generator is chosen, the output is going to be a tree of random data, which the modelling system will then bind to the defined random parameter. To describe this case we therefore need:

$F \stackrel{\text{def}}{=} \{f_1, \dots, f_S\}$ set of all S available scenario generators

$f_i$ shorthand notation to identify the choice of a scenario generator, $1 \leq i \leq S$

`

$n_i$ number of parameters needed by scenario generator *i*

$\pi_i \stackrel{\text{def}}{=} \{\pi_1, \dots, \pi_{n_i}\}$ parameters needed by scenario generator *i*

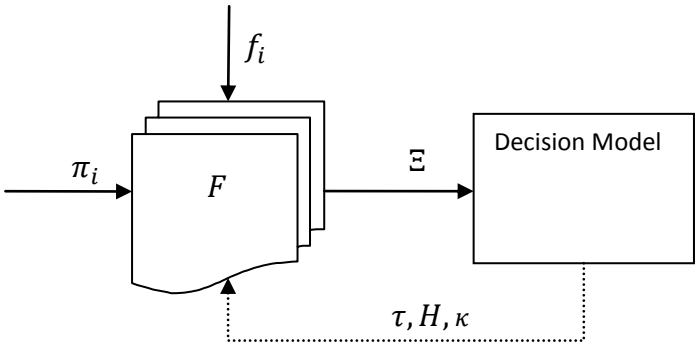



**Figure 24 Abstract representation of a SG library, single random parameter**

The dashed arrow at the bottom means that, in case the chosen scenario generator $f_i$ needs one of the parameters that are well known (dimension of the random parameter, historical data and tree structure), these can be communicated from the model.

### *Multiple scenario generators – Multiple random parameters*

To face this case, we have to introduce some more notation, which relates to the fact that the random vector previously defined as $\xi(\omega)$ can, in an algebraic perspective, be decomposed in various random parameters $\rho(\omega)$, grouped by their meaning in the model (i.e. for a financial model: asset prices, interest rates, mortality rate, cash incomes, …).

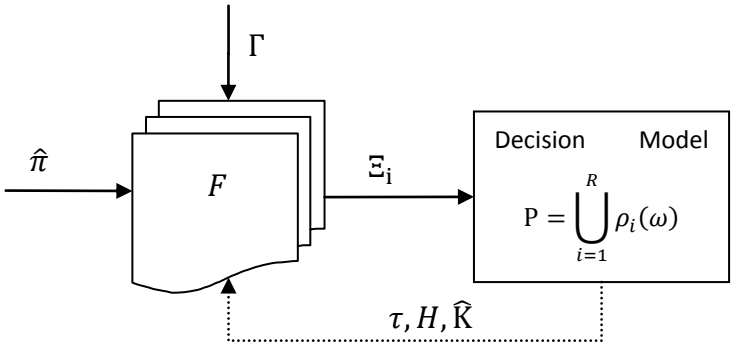
$$\mathrm{P} = \bigcup_{i=1}^{R} \rho_i(\omega) \qquad \textbf{4.15}$$

These random parameters' uncertainties might be appropriately modelled with different stochastic processes, therefore, in our perspective, the parameters values are generated via different scenario generators. We need therefore to assign a mapping between each random parameter and the scenario generator to be used for it. In case of scenario generators that can generate multivariate distributions that span for more than one of the defined random parameters, this mapping needs to be *n-to-1*. We denote with the letter $\Gamma$ the map between random parame-

`

ters and scenario generators: $\Gamma: \mathrm{P} \rightarrow \mathrm{F}$. Note that this map is not injective nor surjective: all random parameters have to be assigned, not all the available scenario generators need to be used and more than one random parameter can be assigned to one scenario generator.

To shorten the notation, the accent denote in $\pi, H, \mathrm{K}$ the fact that they are respectively: the set of sets of all parameters needed by the scenario generators chosen in $\Gamma$, the set of historical data sets for all random parameters in the model and the set of dimensions for all random parameters in the model. See Figure 25 for a graphical representation of this abstraction.




| | |
|---|---|
| $\mathrm{P}$ random parameters in the decision model | $F$ set of all scenarios generators |
| $\Gamma$ map random parameters-scenario generators | $\tau$ scenario tree shape |
| $\Xi_i$ generated scenario trees | $\hat{H}$ historical data |
| $\hat{\pi}$ set of all parameters needed by all scenario generators in $\Gamma$ | $\hat{\mathrm{K}}$ dimensions of random vectors |

**Figure 25 Abstract representation of a SG library, multiple random parameters**

## 4.5 SG Library: an IT perspective

The abstraction framework provided in the previous section proved to be essential in the design of a software system to support stochastic programming problems. Most of the identified entities translate directly into classes, the building blocks of a program following the object oriented paradigm however the design of a software system based on the defined abstraction model requires more information; more specifically, *meta-information* about various entities has to be exchanged. This sec-

`

tion presents the steps and the choices available in this respect, focusing on the ones that have been implemented in the current software system.

Overlaying Figure 25 on the software tools building blocks, the flow of information between the two systems appears to be not uni-directional (see Figure 26 below):

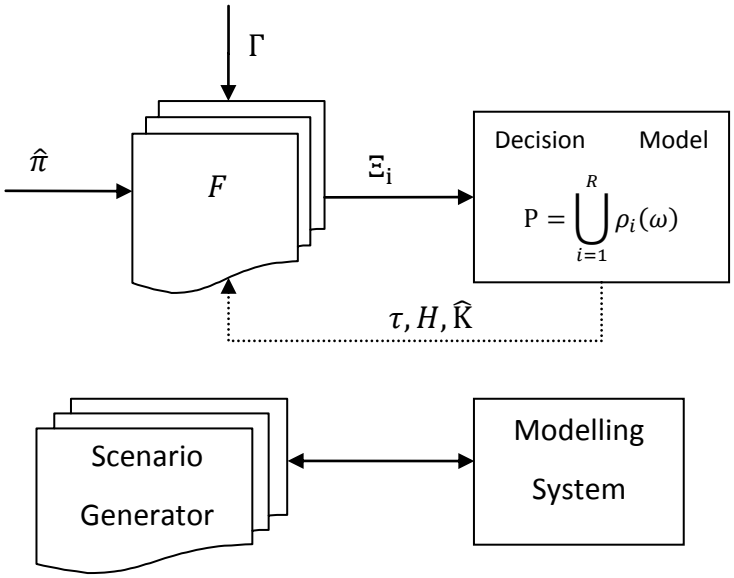



**Figure 26 IT view of Scenario Generation**

The first problem arises already with $\Gamma$, the mapping between random parameters and scenario generators: being defined as $\Gamma: P \rightarrow F$, it involves two entities that are known to two separate systems. P, the list of random parameters are known to the modelling system once it parses the decision model, while F, the set of available scenario generators, is known to the scenario generation library at runtime. The functionality we wanted was to allow the modeller to define the mapping in the model itself, as he would do with a normal data source, hence the choice to communicate F to the modelling system.

### *Metadata*

F, in this perspective, is not a set of functions, but a set of function *descriptions*; these descriptions must include the list of parameters the scenario generator needs, together with their descriptions. To summarize, indicating with the box the

`

fact that we are referring to meta-information, the entities that need to be known for the modelling system to be able to allow the definition of $\Gamma$ is $\boxed{F} = \left\{\boxed{f_1}, \dots, \boxed{f_S}\right\}$ and $\boxed{\Pi_i} = \left\{\boxed{\pi_1}, \dots, \boxed{\pi_{n_i}}\right\}, i = 1, \dots, S.$

The correspondence between entities and classes is the following:

$\boxed{F}$ ⟺ array of `SGInformation` $\boxed{\Pi}$ ⟺ array of `Parameter`

$\boxed{f}$ ⟺ `SGInformation` $\boxed{\pi}$ ⟺ `Parameter`

Obviously, the list of parameters for a particular scenario generator is logically part of its meta-information; the class `SGInformation` therefore contains a name, a description and an array of objects of class `Parameter`, together with other useful information. Each parameter, in turn, is represented by a name, data type, description and a number identifying its position on the command line (see next section for this last item); a UML class diagram is presented in Figure 27 below.

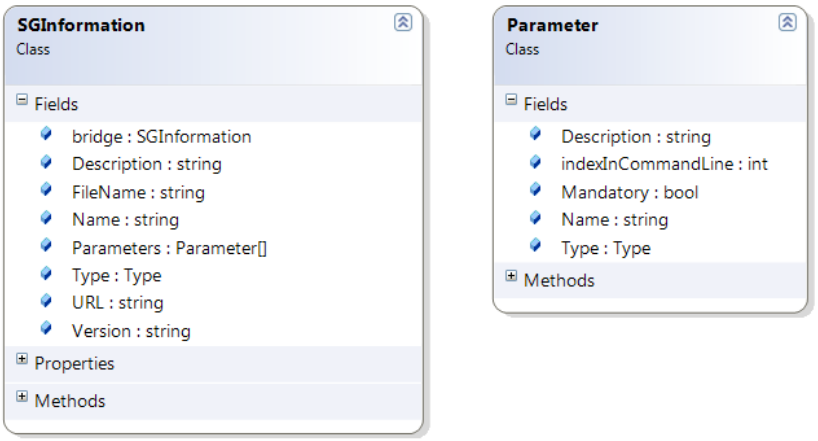



**Figure 27 UML schema of metadata**

## *Functional view*

Generating the scenarios for an SP problem using the system presented, can be seen as a three phases process, which imply some functionalities in the entities involved (modelling system or scenario generator). This (somewhat simplified) analysis leads to the definition of an interface between the modelling system and scenario generators, which is now implemented and fully working.

`

### 1) Discovery / Self-description

Discovery is the process by means of which the system becomes aware of the available scenario generators; it requires the scenario generators to be self-descriptive, functionality that makes use of the data structures defined previously in this section.

| Scenario Generator | Modelling System |
|---|---|
| **`getName`** returns the name of the scenario generator | **`obtainSGList`** gets the list of scenario generators in a specified folder. Internally, besides the obtained `SGInformation`, stores the location of the file where the SG is implemented |
| **`getDescription`** returns the description of the scenario generator | |
| **`getParameters`** returns the list of parameters needed by the scenario generator | |

**Table 13 Functions used for SG discovery**

This functionality can be seen at work in Figure 28 below. The data displayed in the window is generated automatically at application start-up, which is when the discovery process is executed. To the left, the list of scenario generators; the names are obtained calling the function `getName` of the libraries which implement the interface. Once one is selected, the right side of the window is then populated: the *Description* textbox displays the text returned by calling the function `getDescription` while the *Parameters* section displays the information regarding all the entities received by calling the function `getParameters`.

`

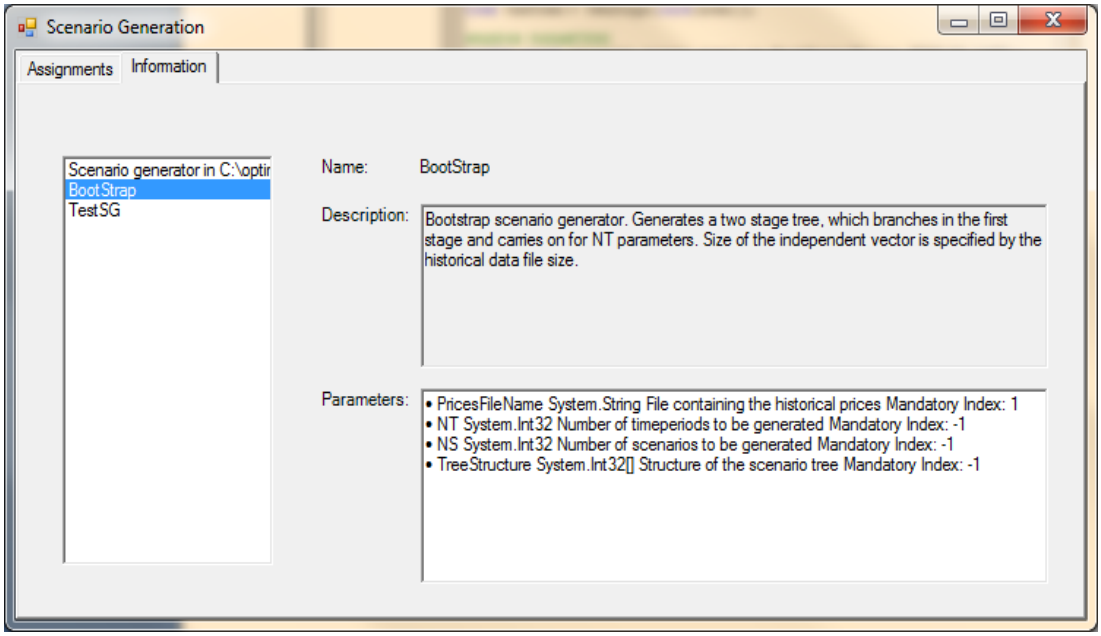



**Figure 28 SG discovery screenshot**

## 2) Mapping / Interpreting

The second step is to create the map between random parameters and scenario generators. At this point, all the information needed is in the modelling system; the decision model defines the scenario generators assigned to the random parameters through specific syntactic constructs, and specifies their parameters. The syntax is briefly introduced in Section 4.6; here it is important to notice that, due to the run-time availability of $\boxed{F}$, the modelling system can check if a scenario generator with the name specified in the decision model actually exists, and if the parameters that are passed to it correspond to its expectations.

Once the parsing is done successfully, storing the mapping is trivial and the modelling system can switch to the next phase, which is the model generation part.

## 3) Generation of the scenario trees

For each mapping $\gamma \in \Gamma$, the system has to call the specified scenario generator with the indicated parameters. The usage of scenario generators that are *unknown* to the modelling system has two issues: firstly it is impossible for the system to perform a validation of the parameters passed to the specific SG and secondly the amount of data passed might be large (typically, a high volume of historical data can be required). These two issues make unadvisable the use of function calls of the

`

form `generate(π)`: if an error in the parameters is present (i.e. type mismatch), the function would just terminate with errors, and the scenario generator has to include complicated error handling code. Thus a different approach has been chosen: the scenario generators modules are required to expose the four functions:

```
bool setParameter(string name, object value);

bool Generate();

string getMessage();

ScenarioTree<double[]> getScenarioTree();
```

The behaviour is then the following:

1) For each parameter $\pi_i \in \pi$, the modelling system calls the function `setParameter` in the SG module, with the values available to the modelling system.
    a. If the returned value is true, then go to 1
    b. If it is false, break and call the function `getMessage`, that will return an error message
2) The modelling system calls `Generate`
    a. If the returned value is true, go to 3
    b. If it is false, break and call the function `getMessage`, that will return an error message
3) Call the function `getScenarioTree`, which returns the generated data in an appropriate data structure

### 4) Enumeration

One aspect remains to be covered, and has not been fully developed in the implementation yet: a scenario tree is generated by each scenario generator in the mapping $\Gamma$, each corresponding to one or more random parameters' realization along the dynamic structure of the decision model. Considering that the scenario tree structure $\tau$ is specified in the decision model, this implies that all these event trees inherit the shape defined by it; the final decision problem is therefore a cross product of all event trees, and the user has no direct control over it.

`

One possible approach is to allow the scenario generation library, which will have then to become more than just a collection of interfaces and data types, to include specific tree enumeration routines, possibly scenario reduction routines. This will allow the library to pass back to the system a single tree $\Xi$ which has the structure $\tau$ specified by the model.

Another possible approach is similar to the previous one, except that the shape of the returned tree is determined by the tree recombination or scenario reduction routines of the SG Library. The tree needs not to be defined in the decision model; this approach has the advantage that scenario reduction routines, or the scenario generator itself, could decide how complex the tree should be to represent the underlying distribution with a certain precision.

A third approach, which is the one at the moment implemented, is that all the generated trees $\Xi_i$ are communicated back to the modelling system that creates the recombined tree doing a simple cross product of the generated trees $\Xi_i$, thus . This has the advantage of simplicity, but the sizes of the created trees increase very rapidly with the number of random parameters.

### *Implementation*

This section presents some details of the implementation work done by the author, taking into account all what examined so far.

The system is written using a .NET compliant language; therefore all the data structures and the interface have to be of that same kind. The scenario generators need to implement the interface `SGModule`, which allows the system to recognize them, get the meta-information and finally exploit their functionalities. The following figure shows the interface, as implemented in a working system by the author thesis.

`

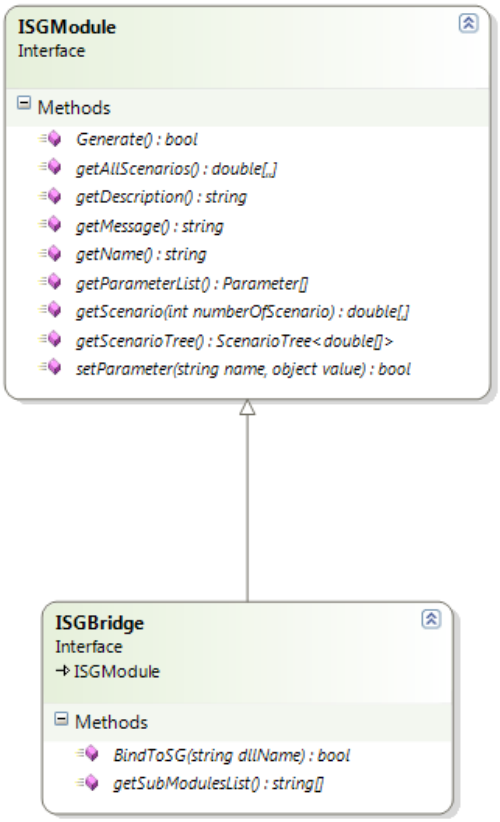



**Figure 29 Scenario Generator Interfaces**

## *ISGModule*

The interface that has to be implemented by any scenario generator to be usable by SPInE is specified as follows:

```
string getName();
```

Return the name of the scenario generation module, to be displayed by the system at runtime. Note that it is the identifier that the user will have to use to identify the scenario generator in his models, so it has to be unique.

```
string getDescription();
```

Return a short text describing the SG, displayed by the system to guide the user.

```
Parameter[] getParameterList();
```

Return the list of the Parameters needed to the scenario generator. The list is in form of an array of Parameter, where the class parameter is specified in Figure 27.

```
bool setParameter(string name, object value);
```

`

Set the parameter corresponding to name to the specified value. If the value is of wrong type, throws TypeException.

```
string getMessage();
```

Return the internal error message, if any. It is called by the system after a failed generation attempt to display the details of the error to the user.

```
bool Generate();
```

Generate the scenarios, returns true if successful, false otherwise. It can fail for a number of reasons, the most common of which is the parameters being not well set.

```
double[,] getScenario(int scenarioNumber);
```

Get a single scenario. Parameter scenarioNumber is the 0-based index of the desired scenario. The returned array is to have the following dimensions [time, indipendentvariables]. Optional.

```
double[, ,] getAllScenarios();
```

Get all the scenarios. The returned array is to have the dimensions [time, scenario, index]. It is used if getScenarioTree() is not present or returns null.

```
General.ScenarioTree<double[]> getScenarioTree();
```

Get all scenarios in the ad hoc datastructure provided in SGHelper.dll. Preferred method of comunicating scenario data.

### *ISGBridge*

The second interface in the figure, which inherits from `ISGenerator`, has the name `ISGBridge`. This is an extension to the system, which allows users to use scenario generators that for various reasons cannot implement `ISGModule` (i.e. they are not written in a .NET compliant language). This is achieved simply defining a bridge (which implements `ISGBridge`) that has the ability to link to the non-conform sub modules. An example of such a technology has been implemented, that allows the system to connect to scenario generators built in MATLAB (see Appendix B). A description of functions defined in this interface follow:

`

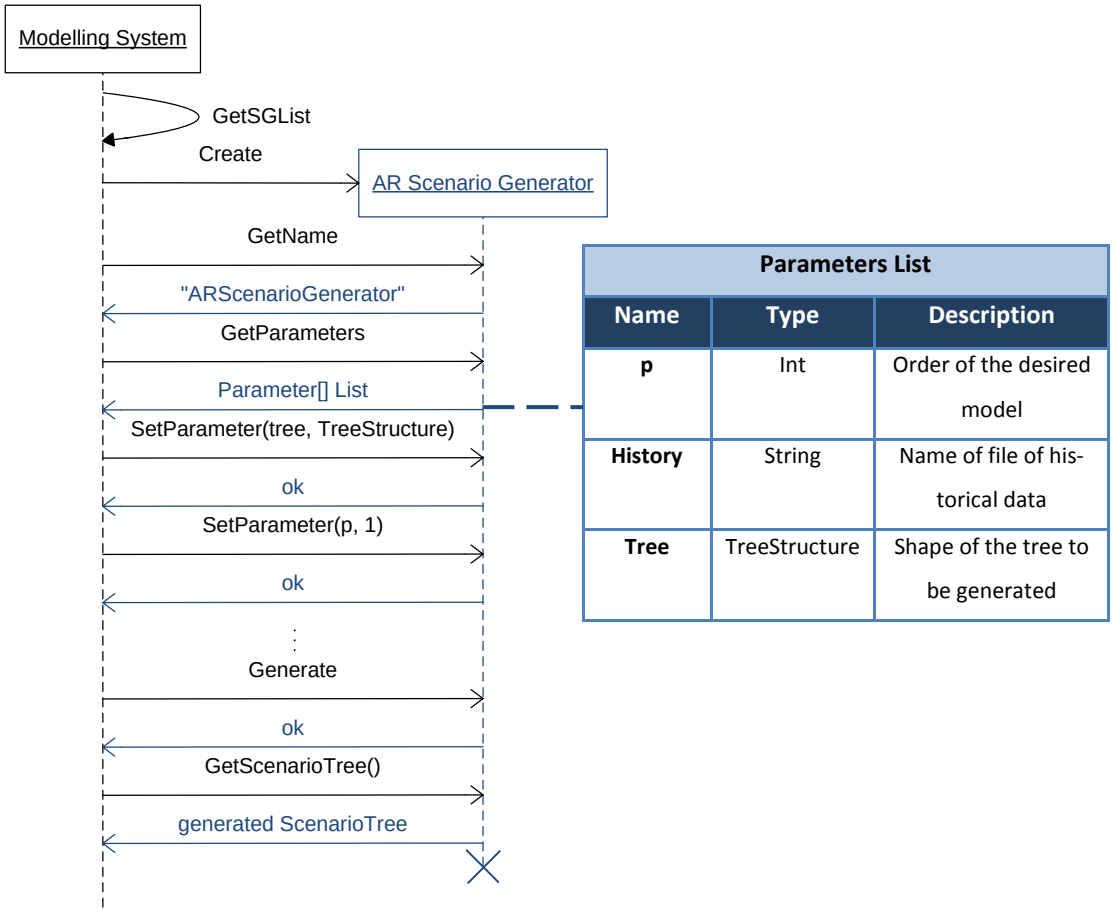
```
bool BindToSG(string dllName);
```

Bind the bridge to a module. After a call to this function, with as a parameter the name of a scenario generator module which is accessible by the bridge, the bridge forwards all the ISGModule calls which inherits to that specific SG.

```
string[] getSubModulesList();
```

Return the names of the modules that can be parsed by the bridge. SPInE will try to parse those libraries through the bridge in successive calls.

Figure 30, below, shows the sequence diagram of a typical session of scenario generation guided by the modelling system, supposing that the decision model contains a random parameter that is mapped to an SG module which is called ARScenarioGenerator and that the correct parameters are passed to it.




| Parameters List | | |
|---|---|---|
| **Name** | **Type** | **Description** |
| **p** | Int | Order of the desired model |
| **History** | String | Name of file of historical data |
| **Tree** | TreeStructure | Shape of the tree to be generated |

**Figure 30 Sequence of interactions MS-SG**

`

The table to the right of the figure shows the parameters list that is communicated back from the scenario generator, meaning that these are the parameters that the chosen SG will need before it can successfully start the generation procedure. After the execution of the shown functions, the modelling system can generate the instance of the chosen model. It is just an example of a successful execution of the system; many possible errors have to be taken into account, and the real data structures are here not fully specified.

## 4.6 Language constructs for the SG library

Language extensions to SAMPL language specifications to support the SG library have been developed; they regard the declaration of random parameter that now includes a string specifying the scenario generator to be used and its parameters. The formal language specification is as follows: random parameter declarations have a list of optional attributes, optionally separated by commas.

**random param** *name* $alias_{opt}$ $indexing_{opt}$ , $attributes_{opt}$ , $sgspecification_{opt}$;

In our implementation we have introduced the last optional attribute, namely *sgspecification*; the formal definition of a random parameter in SAMPL is reported just for that part, the interested reader can refer to Appendix C for a full definition.

*sgspecification:*

*sg name (*$parameterlist_{opt}$*)*

> *name:* a string identifying a scenario generator in the library
>
> *parameterlist:* comma separated list of parameters, the interpretation of which is up to the external sg module specified by *name*

This specification differs from any existing AMPL construct as the correctness of the model can be resolved just at runtime; the names of the available scenario generators, as well as their parameters, are indeed discovered by the system at each execution. If the random parameter specifies in the *name* part of the *sgspecification* construct a scenario generator that is not available to the modelling system, the error will be detected while parsing and reported. Where the required SG is available, an inconsistency of type or number of its parameters is detected by the model-

`

ling system itself, that is at that point aware of the data requirements of each plug-in in the SG library. Incongruent or wrong *meaning* of the data is conversely detected by the scenario generator itself and (hopefully) communicated back to the modelling system which will report it to the user.

To further describe the behaviour of the system, the reader is put in the hypothetical situation of the system correctly recognizing a scenario generator that returns *BrownianMotion* as a name and the following to the system meta-information related queries:

| | Parameter 1 | Parameter 2 | Parameter 3 |
|---|---|---|---|
| **Name** | TreeStructure | NI | HistDataFile |
| **Type** | ScenarioTree | int | string |
| **Description** | Shape of the tree to be generated | Number of independent variables | File containing the historical data, CSV |
| **IndexInLine** | -1 | -1 | -1 |

**Table 14 Scenario generator example**

A formally correct SAMPL declaration that makes use of the above scenario generator is:

```
random parameter price{product, time, s} sg BrownianMotion
("..\Prices_FTSE100_first80.csv");
```

It is worth noticing that out of the three parameters required by the scenario generator, the statement provides just one, namely "..\Prices_FTSE100_first80.csv". This is possible because some parameters are well known by the modelling system and therefore can be omitted by the modeller. These are the tree structure, the dimension of the current random vector, the members of its indexing set and so on; the scenario generator can refer to these known parameters specifying predetermined names, which list is available in the scenario generator programmer's manual (Valente, 2010). These parameters are therefore not needed in the sgspecification string, and this can be noticed by the fact that the IndexInLine property of those parameters, which identifies their position in the SAMPL sgspecification string, has the value minus one.

`

## 4.7 Conclusions

Devising the procedures to generate forecasts of random parameters is a fundamental step in the process of implementing an SP model; this is known as Scenario Generation. There are many methodologies to do so but every problem to be modelled can have its unique procedure for generating random parameters; it seemed therefore a logical necessity to have the means to integrate the scenario generators into the decision model. In this chapter we have described various methodologies used for scenario generation and have illustrated the techniques to seamlessly incorporate scenario generators into the modelling system (Sections 4.4 and 4.5).

`

# Chapter 5 A workflow approach to the investigation of SP models

Decision making under uncertainty by applying the stochastic programming approach is inherently a complex task. The problem owner must come up with an appropriate decision model, a task which is difficult as such; furthermore, he has to specify estimates of the distributions of the random parameters and discretize them, balancing the fineness of the discretization with the resulting computational difficulty. Due to these modelling and computational challenges, making all these modelling choices is not obvious, leading therefore to a strong need of evaluating the performance of the chosen models/techniques in respect to the real world problem. This we call *performance evaluation/investigation,* and involves the use of different techniques depending on the application domain, the entity under test and the available data. Most of these testing procedures can be schematized via an appropriate organization of a *few* kinds of actions, applied to different entities. Taking into account this consideration, the representation of these procedures can make use of workflows and the implementation can follow the same paradigm. In this chapter we focus on the definition and application of workflows for investigation of SP problems. A brief introduction to workflows is given in section 5.1, which is literature review aimed to introduce the concepts needed to formalize the workflows as used in this thesis; a bottom-up description of the activities which compose an investigation workflow follows in section 5.3. Section 5.4 gives examples of the use of workflows to represent real life investigation and decision evaluation problems.

The author's contributions are the idea of applying the workflow formalism to the SP modelling process, in particular to the simulation/validation stage, and the architecture and choice of the atomic operations. Moreover, the development (not yet finished) of the modules and of the controlling GUI, together with the collection, understanding and categorization of simulation techniques/procedures are part of the present efforts of the author.

`

## 5.1 Workflows and workflow management systems

Workflow is a word that has different meanings, depending on the context in which it is used. The first use of it can be traced to the beginning of the last century, where it represented a study to achieve rational organization of work in manufacturing processes. The *flows* under study at these early times were mainly flows of mass and energy (physical resources), although conceptual models of queuing systems and even information flows were already starting to be developed. Nowadays, a workflow is a pattern of activity enabled by systematic organization of resources, roles and flows into a work process that can be documented, learned and partially quantified. This is achieved representing the process through a series of *activities*, or *atomic operations*, interconnected by *resource* or *information flows* and whose sequence and coordination is determined by *control flows.* To support these efforts in rationalization, a formalism has to be introduced, which could capture the process structure in the real world and translate it, with minimum ambiguities, into IT system requirements. Despite the interest in the area, there is still little consensus about its conceptual and formal foundations: each WFMS implement its own language and features, with various syntactic restrictions, and studies have been made to understand to what extent these differences are fundamental in nature. A possible formalization is given below (van der Aalst et al., 1994).

### *Definitions*

**Task** Piece of work to be done by one or more *resources* in a time interval. It is considered atomic (not divisible in smaller tasks). The requirements are given in terms of *resource classes*.

**Resource** Any asset able to carry a work unit (or *task*). A *resource* doing a *task* is occupied for the time interval needed to perform it, but may be assigned to two or more *tasks* at the same time.

**Resource class** A set of *resources*.

**Document** The input or output of a *task*, as far as it is relevant to the *workflow management system*.

`

**Resource Manager** Entity which controls the allocation of *resources* to *tasks*.

**Procedure** Partially ordered set of *control activities*, *tasks*, *resource classes* and (sub)*procedures*. The ordering models the order in which the composing entities have to be performed.

**Control activity** Specifies the routing of the work within a *procedure*, and the synchronisation of *tasks*.

**Job** A process modelling the execution of an amount of work according to a given *procedure*. It is basically an "instance" of a procedure. It is characterized by a sequence of *states*.

**Job state** Contains all the relevant information of the history of the *job* at that moment, a job identification and some *job attributes*

**Job attributes** Are used to determine the routing of a *job*

**Workflow** is a partially ordered set of *jobs*

**Workflow management system**: a software that manages *workflows*. It provides the following functions: (i) definition of *tasks*, *procedures* and *jobs* (ii) processing of the information that is needed to perform the *tasks* which compose the *jobs* (iii) the management of *resources* (iv) routing of job information between *procedures* and *resources*.

A common approach in the literature to formalize these concepts and to model workflows and workflow management systems is through Petri nets, in particular Petri nets extended with 'colour' and 'time'. For an introduction to Petri nets, see (Petri & Reisig, 2008), cured by Carl Adam Petri himself. These extensions to Petri nets allow to assign a 'colour' to a token, which can be used to identify it, and 'time' constraints to transitions (Van Hee, 1994). Control activities will be modelled by *transitions*, jobs are represented by *tokens*.

A *task* can be modelled as shown in Figure 31 below:

`

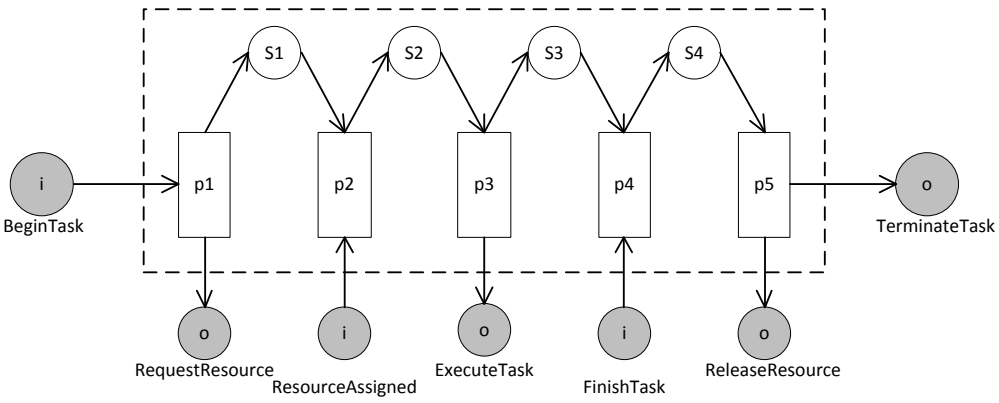



**Figure 31 Petri net model of a task**

where the empty circles represent places (or states), the greyed circles are connectors which allow inter-system communication (the letters i and o represent the direction) and the rectangles represent transitions (or procedures). A task communicates to a procedure (via the connectors **BeginTask** and **TerminateTask**), to the resource manager (via **RequestResource**, **ResourceAssigned** and **ReleaseResource**) and to a resource (via **ExecuteTask** and **FinishTask**). Every task can be modelled in this way, which captures the mechanism by which a task is included in a procedure and by which it can acquire the resources it needs to execute its work item and make it available to the system when it doesn't need it anymore. In the same way, it is possible to model the procedures, the resource manager and the whole WFMS (WorkFlow Management System). Going in further details on these steps is well beyond the scope of this thesis, in which another, less descriptive but easier formalization is used to describe the workflows. The formalization is shown in section 5.2.

In their broad and general scope, workflows are used to represent business processes, which are supported by the WFMS. The IT structure to represent and control the actual processes is often organized in a similar way, so that a modification of the business process leads to changes in the IT system which can be inferred directly from the changes in the process itself. In recent times, considerable developments have taken place in IT based MFMS. These developments are well described in (Khoshafian & Buckiewicz, 1995), and lead to substantial formalization and implementation work and significant applications in the management sciences.

`

## 5.2 Chosen workflow formalisation

The kind of processes that the implemented system captures are not very complex at this stage, and a somewhat simple definition of the workflows suffices to describe them. I have chosen the following syntax, which ignores the allocation of the resources altogether, as given in (Kiepuszewski et al., 2000):

**Definition 1** *A workflow $W$ consists of a set of process elements $P$, and a transition relation* $\mathrm{Trans} \subseteq P \times P$ *between process elements. The set of process elements can be further divided into a set of or-joins $O_j$, a set of or-splits $O_s$, a set of and-joins $A_j$, a set of and-splits $A_S$, and a set of activities $A$.*

*The outgoing transitions of or-splits may have predicates assigned to them through a function* $\mathrm{Pred}: \mathrm{Trans} \cap (O_s \times P) \rightarrow Pred$. *Activities may have a name assigned to them through the partial function* $\mathrm{Name}: A \nrightarrow Name$. *Activities without names are referred to as null activities. And-joins and or-joins should have an outdegree of at most one, and-splits and or-splits should have an indegree of at most one, and all activities have an indegree and outdegree of at most one. Finally, we call process elements with an indegree of zero initial items* $(I \subseteq P)$ *and conversely, process elements with an outdegree of zero – final items* $(F \subseteq P)$.

Giving a full semantic to this syntax is out of the scope of this thesis. In this thesis we focus on some elementary workflow facilities, and giving a semantic to these facilities can be done mapping the introduced lexical entities to elementary Petri-nets as shown in (Van Der Aalst & Ter Hofstede, 2000), (Salimifard & Wright, 2001). The formalization introduces the concept of *structured workflow,* which is restricted in a number of ways, but is satisfies all our modelling needs:

**Definition 2** *A structured workflow model (SWM) is inductively defined as follows.*

1. *A workflow consisting of a single activity is a SWM. This activity is both* initial *and* final.

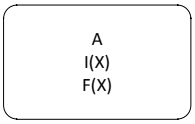



**Figure 32 Single activity SWM**

`

2. *Let X and Y be SWMs. The concatenation of these workflows, where the final activity of X has a transition to the initial activity of Y, then also is a SWM. The initial activity of this SWM is the initial activity of X and its final activity is the final activity of Y.*

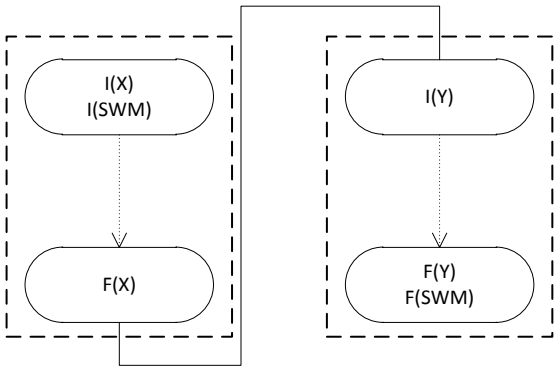



**Figure 33 Concatenation of SWMs**

3. *Let $X_1, \ldots, X_n$ be SWMs and let j be an or-join and s be an or-split. The work-flow with as initial activity s and final activity j and transitions between s and the initial activities of $X_i$ and between the final activities of $X_i$ and j, is then also a SWM. Predicates can be assigned to the outgoing transitions of s. The initial activity of this SWM is s and its final activity is j.*

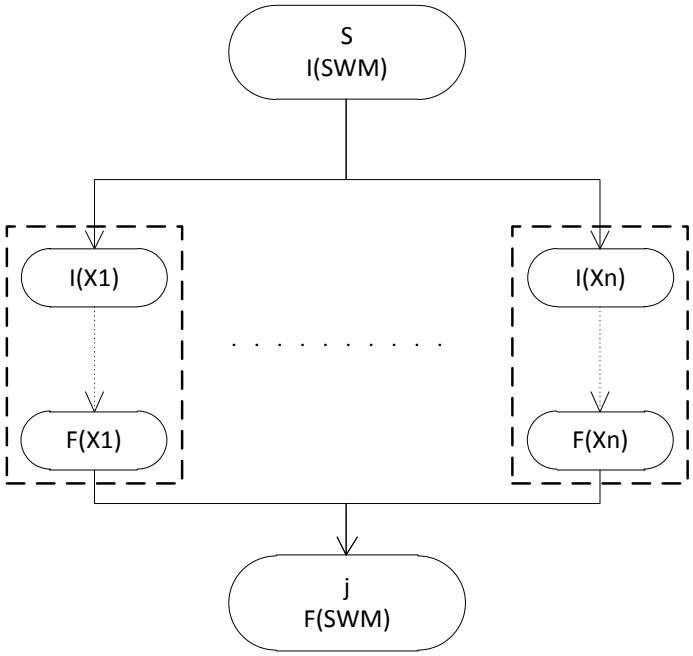



**Figure 34 SWM with join and split activities**

`

4. *Let $X_1, \dots, X_n$ be SWMs and let j be an and-join and s an and-split. The workflow with as initial activity s and final activity j and transitions between s and the initial activities of $X_i$, and between the final activities of $X_i$ and j,, is then also a SWM. The initial activity of this SWM is s and its final activity is j* (see Figure 34).
5. *Let X and Y be SWMs and let j be an or-join and s an or-split. The workflow with as initial activity j and as final activity s and with transitions between j and the initial activity of X, between the final activity of X and s, between s and the initial activity of Y, and between the final activity of Y and j, is then also a SWM. The initial activity of this SWM is j and its final activity is s.*

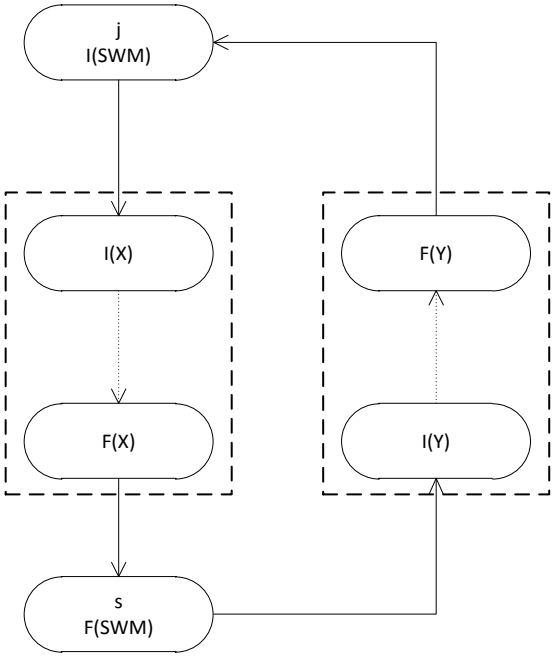



**Figure 35 SWM with loops**

This syntax is proven to be equipotent to many WFMS languages, and it allow the expression of control flow patterns like sequences, parallel splits, synchronization in a formally easy, if not concise, fashion. Lists of control flow patterns are given in (Wohed et al., 2005) and once again, for the scope of this thesis, a comprehensive definition seems verbose and unnecessary.

Because of their peculiar explicative nature, workflows are often represented in a graphical way, assigning different shapes to different conceptual entities. This is the approach that has been chosen to describe the simulation framework design which

`

is the topic of this chapter; the diagrams are simplified, ignoring pure computer science components to better highlight the entities which are related to the stochastic programming area.

Workflow application is a software application with automates a process; they can be developed in any programming language but specialized workflow languages do exist. These language include the XML based XPDL (Workflow Management Coalition, 2010), YAWL (YAWL Foundation, 2010) and BPEL (Organization for the Advancement of Structured Information Standards, 2010) with his various implementations. An alternative approach is to use specialized libraries, like Windows Workflow Foundation, which provide functionalities to control the execution of the activities coded in any .NET language. The choice of this last method to develop the software framework is due to the availability of a re-hostable workflow designer, which can be personalized and embedded in the application, giving the user the ability to graphically create his workflows and relying on the software framework for their execution.

## 5.3 Atomic operations in an investigation framework

Investigation, or decision evaluation, of the parts of an SP model (decision model and its related scenario generators) can be often modelled via a series of interconnected activities *(atomic operations)*. This is due to the repetitive nature, in an abstract point of view, of the procedures involved. The author's method to approach the problem of creating a tool to support investigation was to analyse some methodologies in an operational perspective, identifying their common constituents and recurring types of data that must be handled. The next step was to translate these components into workflow activities that the user of the system can interconnect to create his own simulation/investigation framework. This is an extension to the approach of this work's predecessors (Di Domenica, 2005), which examined the need of the tools (scenario generators and decision model) to perform these analysis, but not a user accessible way to organize them to automate the process. A bottom-up view of the results of this analysis is set out below, and their application follows in the next sections.

`

## General considerations

Normally, the exploitation of a workflow is a two-stage process: firstly the user *designs* the workflow, in the same way a user would write a script file, then the user starts the *execution* of the workflow that has been defined. Due to the particular domain of this workflow meta-model, a three stage procedure has been devised: *design, optimization model parsing* and *execution.* The added phase, namely *optimization model parsing* is not explicitly done by the user; it is instead performed "under the hood" by the system to assist the user during the design phase. More precisely:

***Design*** is responsibility of the user, who places the blocks corresponding to the activities into a drawing surface, connects them and specifies the parameters which are needed to the specific activity.

***Optimisation model parsing*** is an operation done by the system at *design* time, and occurs automatically when the user places some activities on the drawing surface, or when the user specifies some parameters. It provides the system with some information about the model used which can help the user in filling the properties of the remaining activities. It is not related to workflow parsing or validation in any way, as these functionalities are provided by the framework (Microsoft Workflow Foundation) which hosts the execution at runtime; indeed, this functionality is executed by SPInE (the system), which reads the specified model file and returns these quantities to the workflow designer.

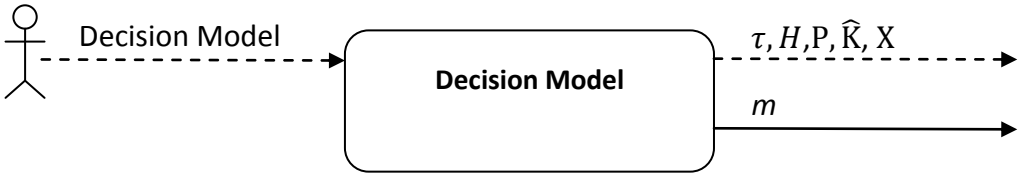
***Execution*** is initiated by the user, and corresponds to the sequential execution of all the activities specified in the workflow.

In the following figures, the dotted lines identifies data passed at design time, the normal lines data which is passed at execution time and the man shaped icon identifies input which is required from the modeller. I follow, whereas possible, the notation introduced in Chapter 4.

`

### 1) Choose decision model

At design time, it allows the choice of the decision model file to be used in the next blocks. It automatically parses the model, getting $\tau, H, \hat{\mathrm{K}}, \mathrm{P}$ and the vector of decision variables $\mathrm{X}$ grouped by algebraic declaration and by stage. This information is communicated, still at design time, to the next connected blocks. At execution time, this block simply passes the algebraic model *m* to the next modules.




$\mathrm{P}$ random parameters in the decision model

$\hat{H}$ historical data

$\tau$ scenario tree shape

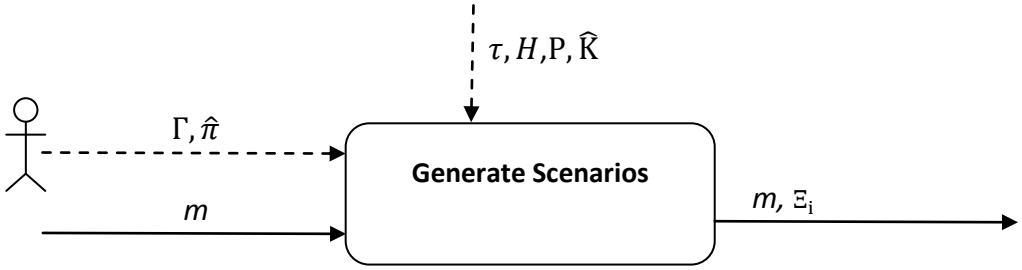
$\hat{\mathrm{K}}$ dimensions of random vectors

*m* algebraic decision model

**Figure 36 Activity 1: Choose decision model**

### 2) Generate Scenarios

At design time, it allows the definition of the mapping $\Gamma: \mathrm{P} \rightarrow \mathrm{F}$ between random parameters and scenario generators, and of $\hat{\pi}$, the set of sets of parameters needed by the chosen scenario generators. In the execution phase, the block generates the set of scenarios needed by the model, accordingly to the mapping defined.




$\Gamma$ mapping between random parameters and scenario generators

$\Xi_i$ generated scenario trees

$\hat{\pi}$ set of parameters required by the chosen scenario generators

**Figure 37 Activity 2: Generate Scenarios**

`

### 3) Solve

At design time, it allows the selection of a solution algorithm and its controlling parameters; at execution time, it generates an instance of the stochastic model m on the sets of scenarios $\Xi_i$ that are provided as an input, solves it and gives the objective function value *Z* and the solution vector $X_*$. The optional input $X_*$, if present, fixes the specified variables to the specified values.

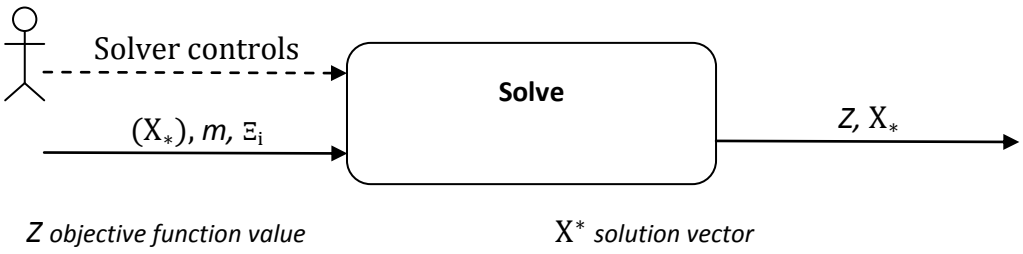



**Figure 38 Activity 3: Generate and solve model**

### 4) Fix Variables

At design time, this activity receives the two vectors of decision variables $X^I, X^O$ (input and output respectively) from two **Decision Model** activities, and the user inputs a injective partial mapping $\Delta: X^I \rightarrow X^O$ that defines which variables in $X^O$ must be fixed, and to the value of which $X^I$. During the execution phase, it receives the solution vector $X^I_*$ and outputs a vector of variables $X^O_*$ that can be used into a **Solve** activity.

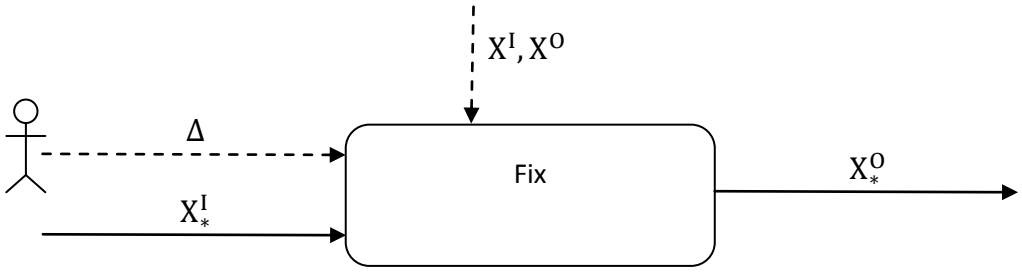
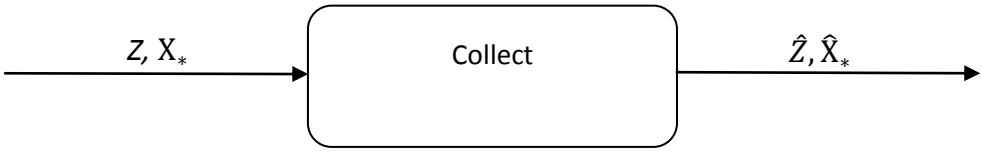



**Figure 39 Activity 4: Fix variables**

`

## 5) Collect Information

At execution time, this activity stores the information it receives as input. If in a loop, it accumulates all the values being passed to it, to then release them to the next activity at the end of the loop. The outputs are therefore the collections $\hat{Z}$ and $\hat{\mathrm{X}}_*$ of objective function values and solution vectors.



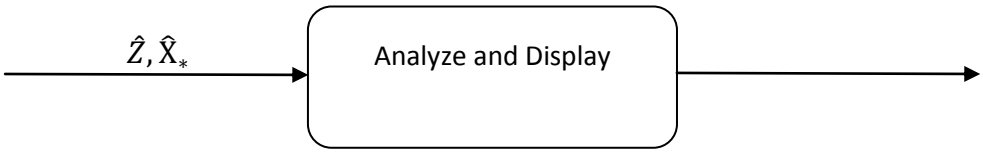

*Z objective function value* $\mathrm{X}_*$ *solution vector*

$\hat{Z}$ *collection of objective values* $\hat{\mathrm{X}}_*$ *collection of solution vectors*

**Figure 40 Activity 5: Collect information**

## 6) Execute Analysis

The result of an investigation is usually an analysis of the distribution of the solutions of a model problem over different runs, or a comparison of solutions obtained using various decision models. This is performed by this activity, where the user chooses at design time which kinds of analysis are to be undertaken and displayed.




$\hat{Z}$ *collection of objective values* $\hat{\mathrm{X}}_*$ *collection of solution vectors*

**Figure 41 Activity 6: Result Analysis**

`

## 5.4 Example Cases

The evaluation of a decision model, a randomness model or a decision is a crucial step to decide their applicability to real life problems. There are various types of decision evaluation techniques; not all of them are expressible using the proposed workflow formalism, and some of them overlap, but they are here reported for completeness.

- **In-sample testing** the most obvious kind of testing, it evaluates how the model/decision under test performs within the original input data
- **Out-of-sample** conceptually similar to the in-sample testing, but it evaluates the behaviour against new data, that is, information that was not used to make the decision under test
- **Back testing** the model/decision is evaluated against past historical data, not used to obtain it, to evaluate how it would have performed in that past scenario
- **Stress testing** the model/decision is tested under "extreme" circumstances (i.e. very low interest rates in financial models, extreme weather events for energy models, multiple breakdowns for fleet allocation problems), preferably not fully captured by the data used to calculate the decision.
- **Scenario analysis** multiple decisions, taken using different scenarios, are evaluated against each other's input data. If used in a stochastic framework, it gives a hint on how much the combination scenario generator/decision model is stable, and whether its performance is dependent on the particular run
- **What-if analysis** evaluation scenarios are explicitly constructed to represent some hypothesis on the future, and the decision is evaluated against them

In general, decision evaluation is a procedure that can be described as a two step process (Di Domenica, 2005), see Figure 42:

**Decision making (ex ante)**

- Select the scenario generators for the decision model

`

- Solve the decision model and obtain the solution vector

**Decision evaluation (ex post)**

- Select the scenario generators for the decision model
- Fix a subset of the decision variables (usually the first stage ones) to the values obtained in the ex ante execution
- Solve the resulting decision model
- Calculate statistical, stochastic and risk measures

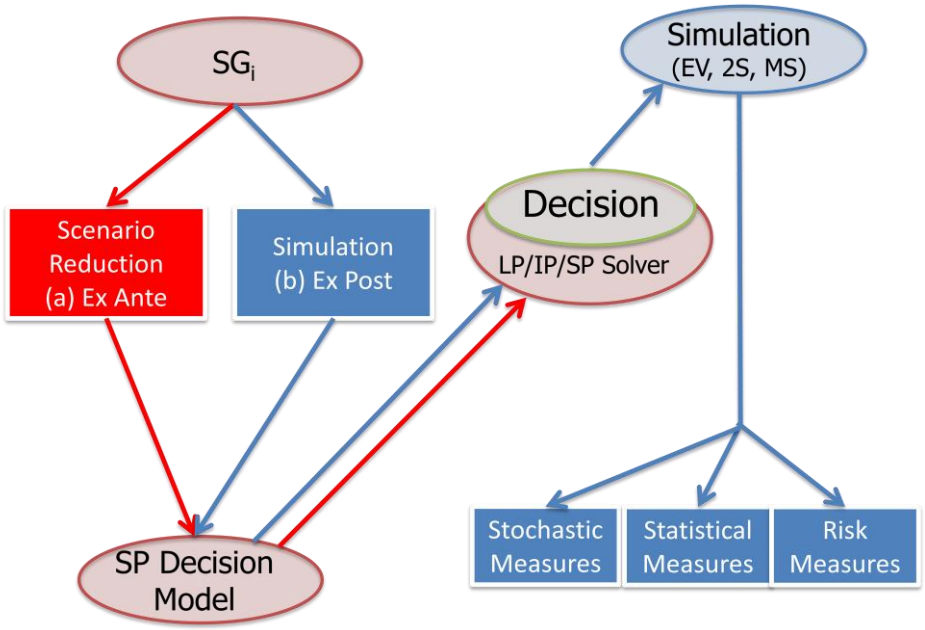



**Figure 42 Decision evaluation schema**

Various refinements and applications of these general guidelines can be devised; some are introduced as use cases below, illustrating how they can be implemented through the workflow based investigation framework the author is proposing.

### 1) SG stability test – In sample (see equation 4.12)

As mentioned in Section 4.3, there are desirable properties that a scenario generator should present, to be practically useful (Kaut & Wallace, 2003). The first one is *in sample stability*, which is an indicator of how the performance of the combination of scenario generator and decision model varies over different runs. It can be captured in a run time workflow as in Figure 43 below.

`

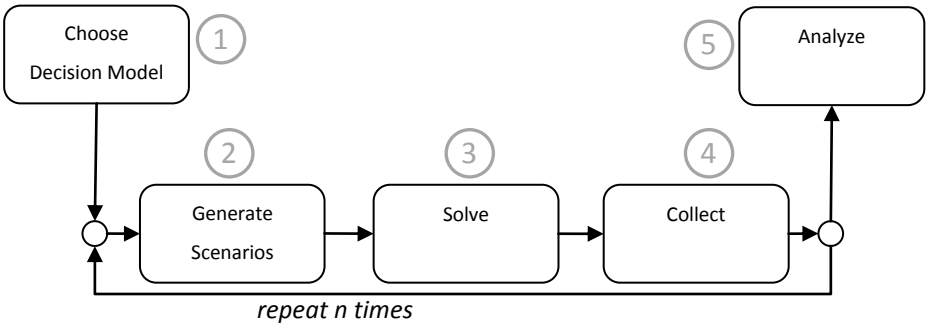



**Figure 43 SG in sample stability workflow**

The user experience while designing the workflow follows; while placing the various blocks, the user is asked to choose:

1. The decision model to be used (a model expressed in SAMPL)
2. The scenario generator to be used, and which parameter in the model is it bound to (a scenario generator part of the library and a random parameter specified in the algebraic model of point 1)
3. The solution method to be used
4. The data to collect (objective function value only or objective function and decision vector)
5. Which kind of analysis to perform (in this case, the distribution of the objective function values)

Information that can be obtained automatically, like the list of random parameters in the model, the list of scenario generators and the list of solvers are obtained by the system at parsing time (part of the design phase), therefore are presented to the user to ease the choice.

### 2) SG stability test – out of sample

A second desired property of a scenario generator is the out of sample stability, which evaluates the performance of the combination scenario generator/decision model over *real world* scenarios, which are historical data or a large scenario set which is believed to accurately capture the underlying uncertainty.

`

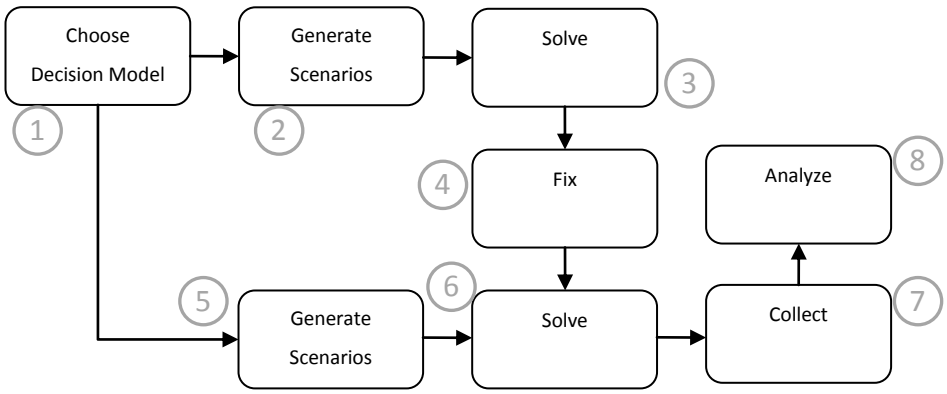



**Figure 44 SG out of sample stability workflow**

In this case the user experience is:

1. The decision model to be used (a model expressed in SAMPL)
2. The scenario generator to be evaluated, and which parameter in the model is it bound to (a scenario generator part of the library and a random parameter specified in the algebraic model of point 1)
3. The solution method to be used
4. The decision variables to be fixed, often just the first stage variables
5. The scenario generator which is believed to be a good representation of real world – can be historical data as a special case
6. The solution method to be used
7. The data to be collected (just the objective function values)
8. Which kind of analysis to perform (in this case, the distribution of the objective function values)

### 3) Out of sample testing / Backtesting

To evaluate a decision model by these paradigms, we evaluate the behaviour of the solutions obtained on data sets which were not used in the original decision making (in case of out of sample testing) or on pure historical data (in case of backtesting). The resulting workflow is drawn in Figure 45, the only difference between the two methods is in that in case of backtesting, the second scenario generator is forced to be historical data.

`

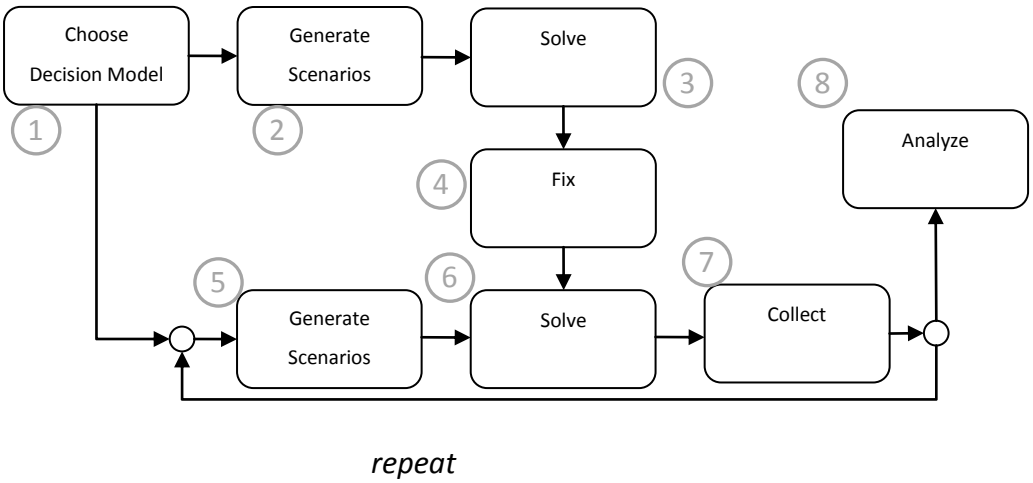



**Figure 45 Out of sample and backtesting workflow**

In this case the user has to choose:

1. The decision model to be used (a model expressed in SAMPL)
2. The scenario generators and their binding to the random parameters in the model
3. The solution method to be used
4. The decision variables to be fixed, often just the first stage variables
5. Scenarios generated with different parameters, or historical data
6. The solution method to be used
7. The data to be collected (often a subset of decision variables)
8. Which kind of analysis to perform

## 5.5 Conclusions

Evaluating the performance of models becomes of paramount importance if the assumptions underlying them are not certain. In Stochastic Programming, this is by definition the case, hence the strong need of simulation and testing. In this chapter we have shown a formalisation of simulation procedures using structured workflows. This formalisation forms the backbone of the further work items recommended to extend the modelling system with graphical facilities to easily represent and efficiently execute model investigation procedures.

`

# Chapter 6 Conclusions

## 6.1 Summary

In this thesis, we investigated the problem of decision making under uncertainty; a number of alternative paradigms are available for such problems, and we set the focus on Stochastic Programming (SP) and Robust Optimization (RO). We reported the mathematical formulation of these modelling approaches, and listed their applications, which comprise many different areas. In SP modelling, the modelling of the uncertain parameters is called scenario generation, and an overview of the most common methods has been given. SP and RO are computational methods and they need to be supported by appropriate software applications. We have therefore proposed a possible architecture of a software tool supporting SP and RO, setting the scope for the rest of the thesis, and linking the modules which are part of the architecture to the steps of SP modelling and solving. The steps we have considered are: *decision modelling, scenario generation, solution,* and *performance evaluation/investigation.*

An overview of the currently available software tools for modelling SP has been given in chapter two, and their peculiarities and shortcomings have been identified. Although it is possible to express and solve SP decision models using software for deterministic optimization by exploiting the deterministic equivalent formulations, it is shown to be unpractical from the modeller's perspective. We have highlighted this fact presenting the same decision model using the syntaxes of AMPL (general purpose algebraic modelling language) and SAMPL (specialized for SP).

In chapter three, we have shown that representing an instance of an SP problem using conventional means is very inefficient memory-wise, due to the extensive data replication that this approach implies. A second drawback of this approach in model instance representation is that it becomes impossible to automatically match and apply existing decomposition algorithms that exploit the peculiar structure of SP problems. The performance gain of such algorithms has been reported in a series of benchmark problems, which have been solved using the decomposition

`

based solver that has been developed by the research team in CARISMA. The link between model types, solution algorithms and the sub-solvers these algorithms require is presented, and the implementation of the automated functionality to choose the best available solution method for the model currently at hand has been briefly presented.

The process of generating the values for the uncertain parameters typical of an SP problem, or *scenario generation*, has been analysed in chapter four. Some desirable properties, like correctness and stabilities are introduced, that make it possible to evaluate the quality of the scenario generator method for a given decision model. Aiming towards the definition of requirements for the scenario generation module of an integrated SP software system, we then presented a series of abstractions of the process of generating scenarios, and of the connections between those models of randomness and the decision model. The result of this analysis has been the basis for our implementation of a scenario generators library, and the consequent language extensions to support it.

In the penultimate chapter, we have introduced the concept of workflow, and some of its applications in various fields. Workflows are organized flows of activities; we have defined a small set of activities that can be used to define investigative/decision evaluation processes. Finally, we have presented some typical investigation frameworks and represented them in our formalism, therefore validating our workflow-based approach.

## 6.2 Contributions of the thesis

In this thesis we have considered a number of important research problems in the domain of decision making under uncertainty. We have also collected very recent research results addressing such problems. An important contribution reported in this thesis is the innovative way these research results have been brought together to design and construct a system which becomes a powerful tool for the OR analysts.

`

The research problems which have been considered and how we have addressed these in our integrated modelling system are listed below.

*(i)* *the modelling of the random behaviour of the model parameters*

This step is inherent to the chosen modelling paradigm, and cannot be avoided. However, the problem owner can benefit by the introduction of the scenario generators library, which collects reusable scenario generation methods, that can be extended due to its modular design

*(ii)* *the interfacing of the decision model with the model of randomness*

This problem is analysed in depth in this thesis through formalization of a scenario generator module and its link to the decision model. Operatively, the proposed scenario generators library interface tackles most of the problems arising in this respect

*(iii)* *the difficulty in formulating and processing SP model instances*

We have identified a number of recently proposed solution algorithms which can process different families of SP models. The combination of the proposed language constructs and the designed software tool can make use of these algorithms using compact instance representation formats, facilitating the formulation and solution procedure

*(iv)* *the requirements for result analysis and performance evaluation through simulation techniques*

The workflow approach proposed in this thesis captures most of the common simulation frameworks. This approach has a well-structured and natural computational structure and avoids the need of programming complicated procedures. It is modular in design and can encapsulate new tools and algorithms, making the most of the information that is available to the system at each stage of the process

`

## 6.3 Future work

Although the experience with the system developed by the author has been positive so far, there are still a number of open problems and chances to develop further. They are listed below, categorized by area of research.

1. *Model generation / Language features*
    a. Rewrite the application as independent column generator (so far it depends on AMPL for generating the scenario sub problems). This would greatly improve the speed of generation, together with system stability;
    b. Add language support for constraints which sums across scenarios (i.e. to allow CVaR-like measures to be assigned to variables with constraints).
2. *Scenario Generation*
    a. Introduce language features to allow AMPL parameters to be used in SG specification statements (depends on 1.a);
    b. Allow multiple scenario generators to be used in multistage models. This is a hard problem, as the combination of scenarios is very problem dependent and so far the author was unable to find a common methodology;
    c. Research into the possibility of introducing already made sampling or clustering blocks in the scenario generator library functionalities, to allow automatic sampling from time series.
3. *Workflow investigation framework*
    a. Complete the development of the described functionalities;
    b. Extend functionalities (by adding other atomic operations) to be able to represent increasingly complicated investigation techniques.

`

# References


[1] Alonso-Ayuso, A., Escudero, L., Garín, A., Ortuno, M. & Pérez, G., 2003. An approach for strategic supply chain planning under uncertainty based on stochastic 0-1 programming. *Journal of Global Optimization*, 26, pp.97--124.

[2] Anon., 2010. *NETLIB library*. [Online] Available at: http://www.netlib.org/lp/index.html.

[3] Ansley, C.F. & Kohn, R., 1986. A note on reparameterizing a vector autoregressive moving average model to enforce stationarity. *Journal of Statistical Computation and Simulation*, 24, pp.99--106.

[4] Ariyawansa, K. & Felt, A.J., 2004. On a new collection of stochastic linear programming test problems. *INFORMS Journal on Computing*, 16, pp.291--299.

[5] Arriola, L. & Hyman, J.M., 2009. Sensitivity Analysis for Uncertainty Quantification in Mathematical Models. In *Mathematical and Statistical Estimation Approaches in Epidemiology*. Springer Netherlands. pp.195--247.

[6] Artzner, P., Delbaen, F., Eber, J.M. & Heath, D., 1999. Coherent measures of risk. *Mathematical Finance*, 9, pp.203--228.

[7] Bai, D., Carpenter, T. & Mulvey, J., 1997. Making a case for robust optimization models. *Management Science*, 43, pp.895--907.

[8] Beasley, J., 1987. An algorithm for set covering problem. *European Journal of Operational Research*, 31, pp.85--93.

[9] Beasley, J. & Jornsten, K., 1992. Enhancing an algorithm for set covering problems. *European Journal of Operational Research*, 58, pp.293--300.

`


[10] Beasley, J., Sonander, J. & Havelock, P., 2001. Scheduling aircraft landings at London Heathrow using a population heuristic. *Journal of the Operational Research Society*, 52, pp.483--493.

[11] Benders, J., 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4, pp.238--252.

[12] Ben-Tal, A. & Nemirovski, A., 1998. Robust convex optimization. *Mathematics of Operations Research*, pp.769--805.

[13] Ben-Tal, A. & Nemirovski, A., 1999. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25, pp.1--14.

[14] Ben-Tal, A. & Nemirovski, A., 2000. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88, pp.411--424.

[15] Bertsimas, D. & Sim, M., 2004. The price of robustness. *Operations Research*, 52, pp.35--53.

[16] Birge, J.R., 1985. Decomposition and partitioning methods for multistage stochastic linear programs. *Operations Research*, 33, pp.989--1007.

[17] Birge, J.R., 1997. Stochastic programming computation and applications. *INFORMS Journal on Computing*, 9, pp.111--133.

[18] Birge, J.R., Dempster, M.A., Gassmann, H.I., Gunn, E.A., King, A.J. & Wallace, S.W., 1987. A standard input format for multiperiod stochastic linear programs. *COAL newsletter*, 17, pp.1--19.

[19] Birge, J.R. & Louveaux, F., 1997. *Introduction to stochastic programming*. Springer Verlag.

[20] Birge, J.R. & Qi, L., 1988. Computing block-angular Karmarkar projections with applications to stochastic programming. *Management Science*, 34, pp.1472--1479.

`

[21] Bisschop, J.J., 1986. A priori Model Reduction and Error Checking in Large-Scale Linear Programming Applications. *IMA Journal of Management Mathematics*, 1, pp.211--224.

[22] Bisschop, J.J. & Entriken, R., 2009. *AIMMS: The modeling system, version 3.9*. Manual. Haarlem, The Netherlands: Paragon Decision Technology.

[23] Bixby, R.E., Ceria, S., McZeal, C.M. & Savelsbergh, M.W., 1998. An updated mixed integer programming library: MIPLIB 3.0. *Optima*, 58, pp.12--15.

[24] Bollerslev, T., 1986. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3), pp.307--327.

[25] Bonami, P. & Lejeune, M., 2009. An exact solution approach for portfolio optimization problems under stochastic and integer constraints. *Operations Research*, 57, pp.650--670.

[26] Box, G.E., Jenkins, G.M. & Reinsel, G.C., 1976. *Time series analysis: forecasting and control*. Holden-day San Francisco.

[27] Brooke, G., Kenderick, D., Meeraus, A. & Rosenthal, R., 2008. *GAMSL a user's guide*. Manual. Washington, DC, USA: GAMS Development Corporation.

[28] Brown, R., 1828. XVII. A brief account of microscopical observations made in the months of June, July and August 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. *Philosophical Magazine Series 2*, 4(21), pp.161-73.

[29] Buchanan, C., McKinnon, K. & Skondras, G., 2001. The recursive definition of stochastic linear programming problems within an algebraic modeling language. *Annals of operations research*, 104, pp.15--32.

`

[30] Casey, M.S. & Sen, S., 2005. The scenario generation algorithm for multistage stochastic linear programming. *Mathematics of Operations Research*, 30, pp.615--631.

[31] Charnes, A. & Cooper, W., 1959. Chance-constrained programming. *Management Science*, 6, pp.73--79.

[32] Choi, I.C. & Goldfarb, D., 1993. Exploiting special structure in a primalâ”dual path-following algorithm. *Mathematical Programming*, 58, pp.33--52.

[33] Colombo, M., Grothey, A., Hogg, J., Woodsend, K., Gondzio, J. & Edinburgh, S., 2009. *A Structure-Conveying Modelling Language for Mathematical and Stochastic Programming*. Technical Report. Edinburgh, Scotland: University of Edinburgh.

[34] Consigli, G. & Dempster, M.A., 1998. Dynamic stochastic programmingfor asset-liability management. *Annals of Operations Research*, 81, pp.131--162.

[35] Dantzig, G.B., 1955. Linear programming under uncertainty. *Management Science*, 1, pp.197--206.

[36] Dempster, M., 1988. On stochastic programming. II: Dynamic problems under risk. *Stochastics*, 25, pp.15--42.

[37] Dentcheva, D., Prékopa, A. & Ruszczynski, A., 2000. Concavity and efficient points of discrete distributions in probabilistic programming., 2000. Springer.

[38] Dentcheva, D. & Ruszczynski, A., 2006. Portfolio optimization with stochastic dominance constraints. *Journal of Banking & Finance*, 30, pp.433--451.

`

[39] Di Domenica, N., 2005. *Stochastic Programming and scenario generation: decision modelling simulation and information systems perspective*. PhD Thesis. London: Brunel University.

[40] Di Domenica, N., Lucas, C., Mitra, G. & Valente, P., 2009. Scenario generation for stochastic programming and simulation: a modelling perspective. *IMA Journal of Management Mathematics*, 20(1), pp.1--38.

[41] Dominguez-Ballesteros, B., Mitra, G., Lucas, C. & Koutsoukis, N., 2002. Modelling and solving environments for mathematical programming (MP): a status review and new directions. *The Journal of the Operational Research Society*, 53, pp.1072--1092.

[42] Dupačová, J., 2002. Applications of stochastic programming: Achievements and questions. *European Journal of Operational Research*, 140, pp.281--290.

[43] Dupačová, J., Consigli, G. & Wallace, S.W., 2000. Scenarios for multistage stochastic programs. *Annals of Operations Research*, 100, pp.25--53.

[44] Dupačová, J., Consigli, G. & Wallace, S.W., 2000. Scenarios for multistage stochastic programs. *Annals of Operations Research*, 100, pp.25--53.

[45] Dupačová, J., Gröwe-Kuska, N. & Römisch, W., 2003. Scenario reduction in stochastic programming. *Mathematical Programming*, 95, pp.493--511.

[46] Efron, B., 1979. Bootstrap methods: another look at the jackknife. *The Annals of Statistics*, 7, pp.1--26.

[47] Efron, B. & Tibshirani, R.J., 1997. *An introduction to the bootstrap*. Chapman & Hall.

[48] Einstein, A., 1906. Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilch. *Annalen der Physik*, 8, pp.549-60.

`

[49] Ellison, E. & Mitra, G., 1982. UIMP: user interface for mathematical programming. *ACM Transactions on Mathematical Software (TOMS)*, 8, pp.229--255.

[50] Ellison, F., Mitra, G.P.C. & Zverovich, V., 2009. *FortSP: A Stochastic Programming Solver*. Manual. London: OptiRisk Systems OptiRisk Systems.

[51] Engle, R.F., 1982. Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica: Journal of the Econometric Society*, pp.987--1007.

[52] Engle, R.F. & Granger, C., 1987. Co-integration and error correction: representation, estimation, and testing. *Econometrica*, 55, pp.251--276.

[53] Entriken, R., 2001. Language constructs for modeling stochastic linear programs. *Annals of Operations Research*, 104, pp.49--66.

[54] Ermoliev, Y. & Wets, R.J., 1988. Stochastic programming, an introduction. *Numerical Techniques for Stochastic Optimization*, pp.1--32.

[55] Escudero, L., Galindo, E., Garcia, G., Gomez, E. & Sabau, V., 1999. Schumann, a modeling framework for supply chain management under uncertainty. *European Journal of Operational Research*, 119, pp.14--34.

[56] Escudero, L.F. & Monge, J.F., 2008. A model for risk minimisation on water resource usage failure. *International Journal of Risk Assessment and Management*, 10, pp.386--403.

[57] Escudero, L., Salmeron, J., Paradinas, I. & Sanchez, M., 1998. SEGEM: A simulation approach for electric generation management. *IEEE transactions on Power Systems*, 13, pp.738--748.

[58] Fábián, C.I., 2000. Bundle-type methods for inexact data. *Central European Journal of Operations Research*, 8, pp.35--55.

`

[59] Fábián, C.I., Mitra, G. & Roman, D., 2009. Processing Second-Order Stochastic Dominance models using cutting-plane representations. *Mathematical Programming*, pp.1--25.

[60] Fábián, C.I. & Szőke, Z., 2007. Solving two-stage stochastic programming problems with level decomposition. *Computational Management Science*, 4, pp.313--353.

[61] Fábián, C. & Veszprémi, A., 2008. Algorithms for handling CVaR constraints in dynamic stochastic programming models with applications to finance. *Journal of Risk*, 10(3), pp.111--131.

[62] Fair, R.C. & Shiller, R.J., 1990. Comparing information in forecasts from econometric models. *The American Economic Review*, 80, pp.375--389.

[63] Fleten, S.E., Keppo, J., Weiss, V.K. & Lumb, H.K., 2009. *Derivative Price Information Use in Hydroelectric Scheduling*. [Online] Available at: http://ssrn.com/abstract=1344725 [Accessed 06 July 2011].

[64] Fourer, R., 1984. Staircase matrices and systems. *SIAM Review*, 26(1), pp.1--70.

[65] Fourer, R., 1996. *Proposed new AMPL features - stochastic programming extensions*. [Online] AMPL LLC Available at: http://www.ampl.com/NEW/FUTURE/stoch.html.

[66] Fourer, R., 1997. Database structures for mathematical programming models. *Decision Support Systems*, 20, pp.317--344.

[67] Fourer, R., 1998. Extending a general-purpose algebraic modeling language to combinatorial optimization: A logic programming approach. *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search: Interfaces in Computer Science and Operations Research*, pp.31--74.

`

[68] Fourer, R., 2001. Model level directives for generating alternative formulations of stochastic programs. In *IX International Conference on Stochastic Programming*. Berlin, Germany, 2001.

[69] Fourer, R., Gassmann, H.I., Ma, J. & Martin, R.K., 2009. An XML-based schema for stochastic programs. *Annals of Operations Research*, 166, pp.313--337.

[70] Fourer, R., Gay, D. & Kernighan, B., 2002. *The AMPL Book*. Duxbury Press, Pacific Grove.

[71] Fourer, R. & Lopes, L., 2009. StAMPL: A Filtration-Oriented Modeling Tool for Multistage Stochastic Recourse Problems. *INFORMS Journal on Computing*, 21, pp.242--256.

[72] Fourer, R., Lopes, L. & Martin, K., 2005. LPFML: A W3C XML schema for linear and integer programming. *INFORMS Journal on Computing*, 17, pp.139--158.

[73] Fourer, R., Ma, J. & Martin, K., 2006. *OSiL: An instance language for optimization*. Technical Report. Evanston, IL: Department of Industrial Engineering and Management Sciences, Northwestern University.

[74] Freedman, D., 1972. *Approximating countable Markov chains*. Holden-Day.

[75] Fritzke, B., 1995. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 7, pp.625-32.

[76] Gaivoronski, A.A., 2006. Stochastic optimization in telecommunications. In M. Resende & P. Pardalos, eds. *Handbook of optimization in telecommunications*. Birkháuser. pp.761--799.

[77] Gassmann, H.I. & Ireland, A.M., 1995. Scenario formulation in an algebraic modelling language. *Annals of Operations Research*, 59, pp.45--75.

`

[78] Gassmann, H.I. & Ireland, A.M., 1996. On the formulation of stochastic linear programs using algebraic modelling languages. *Annals of Operations Research*, 64, pp.83--112.

[79] Gassmann, H. & Schweitzer, E., 2001. A comprehensive input format for stochastic linear programs. *Annals of Operations Research*, 104, pp.89--125.

[80] Gay, D., 2001. Random parameters - a step toward conveniently expressing some stochastic programming problems. In *IX International Conference on Stochastic Programming*. Berlin, Germany, 2001.

[81] Gondzio, J. & Grothey, A., 2009. Exploiting structure in parallel implementation of interior point methods for optimization. *Computational Management Science*, 6, pp.135--160.

[82] Gondzio, J. & Sarkissian, R., 2003. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96, pp.561--584.

[83] Grigoriadis, M.D. & Khachiyan, L.G., 1996. An interior point method for bordered block-diagonal linear programs. *SIAM Journal on Optimization*, 6, pp.913--932.

[84] Haneveld, W.K.K., 1986. Duality in stochastic linear and dynamic programming. *Lecture notes in economics and mathematical systems*, 274.

[85] Haneveld, W.K.K. & van der Vlerk, M.H., 2006. Integrated chance constraints: reduced forms and an algorithm. *Computational Management Science*, 3, pp.245--269.

[86] Hareveley Systems, 1976. *OMNI Linear Programming System: User and Operating manual*. Manual. Denville, N.J., USA: Hareveley Systems Hareveley Systems.

`

[87] Heitsch, H. & Römisch, W., 2009. Scenario tree modeling for multistage stochastic programs. *Mathematical Programming*, 118, pp.371--406.

[88] Henrion, R., Küchler, C. & Römisch, W., 2008. Discrepancy distances and scenario reduction in two-stage stochastic mixed-integer programming. *MANAGEMENT*, 4, pp.363--384.

[89] Higle, J.L., Rayco, B. & Sen, S., 2009. Stochastic scenario decomposition for multistage stochastic programs. *IMA Journal of Management Mathematics*.

[90] Higle, J.L. & Wallace, S.W., 2003. Sensitivity analysis and uncertainty in linear programming. *Interfaces*, 33, pp.53--60.

[91] Hochreiter, R. & Pflug, G.C., 2007. Financial scenario generation for stochastic multi-stage decision processes as facility location problems. *Annals of Operations Research*, 152, pp.257--272.

[92] Holmes, D., 1995. *A (PO)rtable (S)tochastic programming (T)est (S)et (POSTS)*. [Online] Available at: http://users.iems.northwestern.edu/~jrbirge/html/dholmes/post.html [Accessed 26 February 2010].

[93] Høyland, K., Kaut, M. & Wallace, S.W., 2003. A heuristic for moment-matching scenario generation. *Computational Optimization and Applications*, 24, pp.169--185.

[94] Høyland, K. & Wallace, S.W., 2001. Generating scenario trees for multistage decision problems. *Management Science*, 47, pp.295--307.

[95] Hürlimann, T., 1993. *Reference manual for the LPL modeling language*. Institute of Informatics University of Fribourg.

[96] IBM World Trade Corporation, 1976. SH19-1094 *IBM Mathematical Programming System Extended (MPSX/270) Program Reference Manual*. Manual. New York: IBM IBM.

`

[97] IBM World Trade Corporation, 1977. SH19-5014 *MGRW Program Reference Manual*. Manual. New York: IBM World Trade Corporation IBM World Trade Corporation.

[98] Jerrum, M. & Sinclair, A., 1997. The Markov chain Monte Carlo method: an approach to approximate counting and integration. *Approximation algorithms for NP-hard problems*, pp.482--520.

[99] Jobst, N.J., Mitra, G. & Zenios, S.A., 2006. Integrating market and credit risk: A simulation and optimisation perspective. *Journal of Banking and Finance*, 30, pp.717--742.

[100] Jobst, N.J. & Zenios, S.A., 2003. Tracking bond indices in an integrated market and credit risk environment. *Quantitative Finance*, 3, pp.117--135.

[101] Jordan, W.C. & Törnquist, M.A., 1983. A stochastic, dynamic network model for railroad car distribution. *Transportation Science*, 17, pp.123--145.

[102] Kall, P., 1979. Computational methods for solving two-stage stochastic linear programming problems. *Zeitschrift f\Ã¼r Angewandte Mathematik und Physik (ZAMP)*, 30, pp.261--271.

[103] Kallberg, J., White, R. & Ziemba, W., 1982. Short term financial planning under uncertainty. *Management Science*, 28, pp.670--682.

[104] Kall, P. & Mayer, J., 1998. On testing SLP codes with SLP-IOR. *New Trends in Mathematical Programming: Homage to Steven Vajda*, pp.115--135.

[105] Kaut, M. & Wallace, S.W., 2003. Evaluation of scenario-generation methods for stochastic programming. *Stochastic Programming E-Print Series*, 14.

[106] Ketron, 1975. *MPS III DATAFORM: User manual*. Manual. Arlington, USA: Ketron Inc. Ketron Inc.

`

[107] Khoshafian, S. & Buckiewicz, M., 1995. *Introduction to groupware, workflow, and workgroup computing*. John Wiley & Sons, Inc. New York, NY, USA.

[108] Kiepuszewski, B., ter Hofstede, A. & Bussler, C., 2000. On structured workflow modelling. In Bergman, B.W.a.L., ed. *Twelfth International Conference on Advanced Information Systems Engineering*. Stockholm, 2000. Springer Verlag.

[109] Kiwiel, K.C., 1985. *Methods of descent for nondifferentiable optimization*. Springer-Verlag.

[110] Konig, D., Suhl, L. & Koberstein, A., 2007. Optimierung des Gasbezugs im liberalisierten Gasmarkt unter Berucksichtigung von Rohren-und Untertagespeichern. *VDI BERICHTE*, 2018, p.83.

[111] Koutsoukis, N.S., Dominguez-Ballesteros, B., Lucas, C.A. & Mitra, G., 2000. A prototype decision support system for strategic planning under uncertainty. *International Journal of Physical Distribution and Logistics Management*, 30, pp.640--660.

[112] Kristjansson, B., 2001. Optimization Modeling in Distributed Applications: How New Technologies such as XML and SOAP allow OR to provide web-based Services. *INFORMS Roundtable, Savannah, Georgia*.

[113] Kuip, C., 1993. Algebraic languages for mathematical programming. *European Journal of Operational Research*, 67, pp.25--51.

[114] Kusy, M. & Ziemba, W., 1986. A bank asset and liability management model. *Operations Research*, 34, pp.356--376.

[115] Lemaréchal, C., 1978. *Nonsmooth optimization and descent methods*. International Institute for Applied Systems Analysis Laxenburg, Austria.

[116] Lemaréchal, C., Nemirovskii, A. & Nesterov, Y., 1995. New variants of bundle methods. *Mathematical Programming*, 69, pp.111--147.

`

[117] Lindo Systems Inc, 2008. *LINGO Modelling System Version 12.0*. Manual. Chicago IL, USA: Lindo Systems Inc Lindo Systems Inc.

[118] Lobo, M.S., Vandenberghe, L., Boyd, S. & Lebret, H., 1998. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284, pp.193--228.

[119] Lucas, C. & Mitra, G., 1988. Computer-assisted mathematical programming (modelling) system: CAMPS. *The Computer Journal*, 31, pp.364--375.

[120] Maggioni, F., Potra, F.A., Bertocchi, M.I. & Allevi, E., 2009. Stochastic Second-Order Cone Programming in Mobile Ad Hoc Networks. *Journal of Optimization Theory and Applications*, 143, pp.309--328.

[121] Maggioni, F., Vespucci, M., Allevi, E., Bertocchi, M. & Innorta, M., 2007. A gas retail stochastic optimization model by mean reverting temperature scenarios. In *Communications to SIMAI Congress*., 2007.

[122] Makowski, M., 2005. A structured modeling technology. *European Journal of Operational Research*, 166, pp.615--648.

[123] Martinetz, T. & Schulten, K., 1991. A neural-gas network learns topologies. *Artificial neural networks*, 1, pp.397--402.

[124] Maxmimal software, 2002. *MPL Modelling System, release 4.2*. Manudl. Washington DC, USA: Maximal Software.

[125] Messina, E. & Mitra, G., 1997. Modelling and analysis of multistage stochastic programming problems: A software environment. *European Journal of Operational Research*, 101, pp.343--359.

[126] Microsoft Corporation, 2010. *Microsoft Solver Foundation*. [Online] Available at: http://code.msdn.microsoft.com/solverfoundation [Accessed 5 February 2010].

`

[127] MirHassani, S., Lucas, C., Mitra, G., Messina, E. & Poojari, C., 2000. Computational solution of capacity planning models under uncertainty. *Parallel Computing*, 26, pp.511--538.

[128] Mitra, L., 2009. *Scenario generation for asset allocation models*. PhD Thesis. London: Brunel University.

[129] Mitra, G., Ellison, F. & Scowcroft, A., 2007. Quadratic programming for portfolio planning: Insights into algorithmic and computational issues Part II Processing of portfolio planning models with discrete constraints. *Journal of Asset Management*, 8, pp.249--258.

[130] Mitra, G., Lucas, C., Moody, S. & Hadjiconstantinou, E., 1994. Tools for reformulating logical forms into zero-one mixed integer programs: Software tools for mathematical programming. *European Journal of Operational Research*, 72, pp.262--276.

[131] Mulvey, J.M., Vanderbei, R.J. & Zenios, S.A., 1995. Robust optimization of large-scale systems. *Operations Research*, pp.264--281.

[132] Nowak, M.P., Romisch, W. & Wegner, I., 2000. Power management in a hydro-thermal system under uncertainty by Lagrangian relaxation. *Annals of Operational Research*, 100, pp.251-70.

[133] Ntaimo, L. & Sen, S., 2005. The million-variable â□œmarchâ□□ for stochastic combinatorial optimization. *Journal of Global Optimization*, 32, pp.385--400.

[134] Optimization Services, 2008. *Optimization Services (OS)*. [Online] Available at: http://www.optimizationservices.org/ [Accessed 26 February 2010].

[135] Organization for the Advancement of Structured Information Standards, 2010. *Web Services Business Process Execution Language Version 2.0*. [Online] Organization for the Advancement of Structured Information

Standards Available at: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf [Accessed 12 March 2010].

[136] Petri, C.A. & Reisig, W., 2008. *Petri net*. [Online] Available at: http://www.scholarpedia.org/article/Petri_net [Accessed 04 April 2011].

[137] Pflug, G.C., 2001. Scenario tree generation for multiperiod financial optimization by optimal discretization. *Mathematical Programming*, 89, pp.251--271.

[138] Rockafellar, R.T., 1976. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14, pp.877-98.

[139] Roman, D., Darby-Dowman, K. & Mitra, G., 2006. Portfolio construction based on stochastic dominance and target return distributions. *Mathematical Programming*, 108, pp.541--569.

[140] Roman, D., Mitra, G. & Spagnolo, N., 2009. Hidden Markov models for financial optimization problems. *IMA Journal of Management Mathematics*, 21(2), pp.111-29.

[141] Ruszczynski, A., 1986. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical programming*, 35, pp.309--333.

[142] Salimifard, K. & Wright, M., 2001. Petri net-based modelling of workflow systems: An overview. *European journal of operational research*, 134, pp.664--676.

[143] Schultz, G.L. & Meyer, R.R., 1991. An interior point method for block angular optimization. *SIAM Journal on Optimization*, 1(4), pp.583-602.

[144] Schwaiger, K., 2009. *Asset and Liability Management under Uncertainty: Models for Decision Making and Evaluation*. PhD Thesis. London: Brunel University.

`

[145] Scicon Computer Services, 1975. *MGG User guide*. Manual. Scicon Computer Services.

[146] Sen, S. & Kothari, D., 1998. Optimal thermal generating unit commitment: a review. *International Journal of Electrical Power & Energy Systems*, 20, pp.443--451.

[147] Smith, J.E., 1993. Moment methods for decision analysis. *Management science*, 39, pp.340--358.

[148] Soyster, A., 1973. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, pp.1154--1157.

[149] Sperry Univac Computer Systems, 1977. *GAMMA 3.4 Programming Reference*. Manual. Sperry Univac Computer Systems.

[150] Stadtler, H., 2005. Supply chain management and advanced planning--basics, overview and challenges. *European journal of operational research*, 163, pp.575--588.

[151] Thiele, T., 1880. Om anvendelse af mindste Kvadraters Metode i nogle Tilfaelde, hvor en Komplikation af visse Slags uensartede tilfrldige Fejlkilder giver Fejlene en systematisk Karakter. *Vidsk. Selsk. Skr*, 5, pp.381-408.

[152] Tomasgard, A., Audestad, J.A., Dye, S., Stougie, L., van der Vlerk, M.H. & Wallace, S.W., 1998. Modelling aspects of distributed processingin telecommunication networks. *Annals of Operations Research*, 82, pp.161--185.

[153] Törnquist, J., 2005. Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms. In *5th Workshop on Algorithmic Methods and Models for Optimization of Railways*., 2005. Dagstuhl Research Online Publication Server.

`

[154] Valente, P., 2002. *Software tools for the investigation of stochastic programming problems*. PhD Thesis. London: Brunel University.

[155] Valente, C., 2010. *SPInE - Scenario generation manual*. Manual. London: OptiRisk Systems.

[156] Valente, P., Mitra, G., Poojari, C. & Kyriakis, T., 2001. Software tools for stochastic programming: a stochastic programming integrated environment (SPInE). *Department of Mathematical Sciences, Brunel University*.

[157] Valente, C., Mitra, G., Sadki, M. & Fourer, R., 2009. Extending algebraic modelling languages for Stochastic Programming. *INFORMS Journal on Computing*, 21, pp.107--122.

[158] Valente, C., Mitra, G., Valente, P., Zviarovich, V., Poojari, C., Ellison, F. & Di Domenica, N., 2002-2011. *SAMPL/SPInE User Manual*. [Online] Available at: http://www.optirisk-systems.com/manuals/SpineAmplManual.pdf.

[159] Valente, C., Mitra, G. & Zviarovich, V., 2011. *Extendend SAMPL syntax to capture various classes of non-deterministic problems*. Working paper. London: CARISMA Brunel University.

[160] Van Der Aalst, W.M. & Ter Hofstede, A.H., 2000. Verification of workflow task structures: A petri-net-baset approach. *Information systems*, 25, pp.43--69.

[161] van der Aalst, W.M., Van Hee, K. & Houben, G., 1994. Modelling and analysing workflow using a Petri-net based approach., 1994.

[162] van der Vlerk, M.H., 1996-2007. *Stochastic Programming Bibliography*. [Online] Available at: http://mally.eco.rug.nl/spbib.html [Accessed 2010].

[163] Van Hee, K.M., 1994. *Information systems engineering: a formal approach*. Cambridge Univ Pr.

`

[164] Van Hentenryck, P., Lustig, I., Michel, L. & Puget, J., 1999. *The OPL optimization programming language*. Mit Press Cambridge, MA.

[165] Van Slyke, R. & Wets, R., 1969. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, pp.638--663.

[166] Volosov, K., Mitra, G., Spagnolo, F. & Lucas, C., 2005. Treasury management model with foreign exchange exposure. *Computational Optimization and Applications*, 32, pp.179--207.

[167] Wallace, S.W., 2000. Decision making under uncertainty: Is sensitivity analysis of any use? *Operations Research*, pp.20--25.

[168] Wets, R.J., 1974. Stochastic programs with fixed recourse: The equivalent deterministic program. *SIAM review*, 16, pp.309--339.

[169] Williams, H., 1987. Linear and integer programming applied to the propositional calculus. *International Journal of Systems Research and Information Science*, 2, pp.81--100.

[170] Wohed, P., der Aalst, W., Dumas, M., Hofstede, A. & Russell, N., 2005. Pattern-based analysis of the control-flow perspective of UML activity diagrams. *Conceptual Modeling--ER 2005*, pp.63--78.

[171] Workflow Management Coalition, 2010. *XPDL Support and resources*. [Online] Available at: http://www.wfmc.org/xpdl.html [Accessed 15 March 2010].

[172] YAWL Foundation, 2010. *YAWL: Yet Another Workflow Language*. [Online] Available at: http://yawlfoundation.org/ [Accessed 10 March 2010].

[173] Yen, J.W. & Birge, J.R., 2006. A stochastic programming approach to the airline crew scheduling problem. *Transportation Science*, 40, pp.3--14.

`

[174] Yu, C.S. & Li, H.L., 2000. A robust optimization model for stochastic logistic problems. *International Journal of Production Economics*, 64, pp.385--397.

[175] Zenios, S., 2006. *Practical Financial Optimization*. Blackwell Publishers, Ox-ford.

[176] Ziemba, W.T., 2003. The Stochastic Programming Approach to Asset, Liability, and Wealth Management. *The Research Foundation of the Association for Investment Management and Research*.

[177] Zverovich, V., Fábián, C.I., Ellison, F. & Mitra, G., 2009. A computational study of a solver system for processing two-stage stochastic linear programming problems. *Working paper, CARISMA, Brunel University*.

`

# Appendix A Connect a Scenario Generator developed in MATLAB

MATLAB is often the language of choice to prototype and implement mathematical models; in the area of stochastic programming, it is frequently used to implement models of randomness, which we call scenario generators. A direct connection to scenario generators is present in SPInE, but connecting a scenario generator requires the implementation of a .NET interface, which is not achievable in MATLAB. This fact sets the scope to the extension to the interface `iSGenerator` which we named `iSGBridge`, that allows the definitions of a bridge between the interface required by SPInE (`iSGenerator`) and the one implementable by the scenario generator. As first example, the scenario generator library is distributed with a prebuilt bridge that allows developers to easily connect to scenario generators written in MATLAB.

### *Programmer's perspective*

The programmer of a scenario generator in MATLAB is provided with a simplified version of the SG interface, and, to further accelerate the development process, a MATLAB template is provided, which contains all the functions with the correct signatures for the usage with SPInE. It consists of four m files, each representing one function, which have to be implemented by the SG programmer:

**getParameters.m, getName.m, getDescription.m, generate.m.**

The templates are self-descriptive (see inline comments for further reference) and provided below. In the form provided, they implement a scenario generator called "TestMATLABNET", which generates a random numbers populated event tree. The input required from the user is just a floating point number which represents the seed for the random number generator.

`

### *getName*

```
% Template function getName.
%     The function returns a string which is the name of the
%     scenario generator implemented in the module.
%     Note that in MATLAB strings are enclosed in '

function name = getName()

name = 'TestMATLABNET';
```

### *getDescription*

```
% Template function getDescription.
%     The function returns a string which describes the sce-
nario
%     generator implemented in the module.
%     Note that in MATLAB strings are enclosed in '

function description = getDescription()

description = 'Description of the scenario generator to be
passed back to SPInE';
```

`

*getParameters*

```
% Template function getParameters
%    This function returns the parameter list for the SG, in a
%    matrix of the form:
%    nameparam1 type1 description1 mandatory1 index1
%    ...
%    nameparamn typen descriptionn mandatoryn indexn
%
%    where type can be any of the following:
%    int, int[], int[,], int[,,]
%    double, double[], double[,], double[,,]
%    bool, bool[], bool[,], bool[,,]
%    string, string[], string[,], string[,,]

%     System defined parameters are specified by their name,
all the
%     other values defined about them are ignored.
%     As quick reference, the system defined parameters are:
%     NI  (int)
%     NS   (int)
%     NT   (int)
%     TreeStructure (int[])
%     IndepNames   (string[])

function a = getParameters()

% Define parameters

param1 = {'NI', 'int', 'doesnt matter', true, -1};

param2 = {'TreeStructure', 'int', 'doesnt matter', true, -1};

param3 = {'Seed', 'double', 'Seed of the random number genera-
tion', true, 1};

param4 = {'NT', 'double', 'doesnt matter', true, -1};

param5 = {'NS', 'double', 'doesnt matter', true, -1};

param6 = {'IndepNames', 'double', 'doesnt matter', true, -1};

% Pack them into the returned array

a = [param1; param2; param3; param4; param5; param6];
```

`

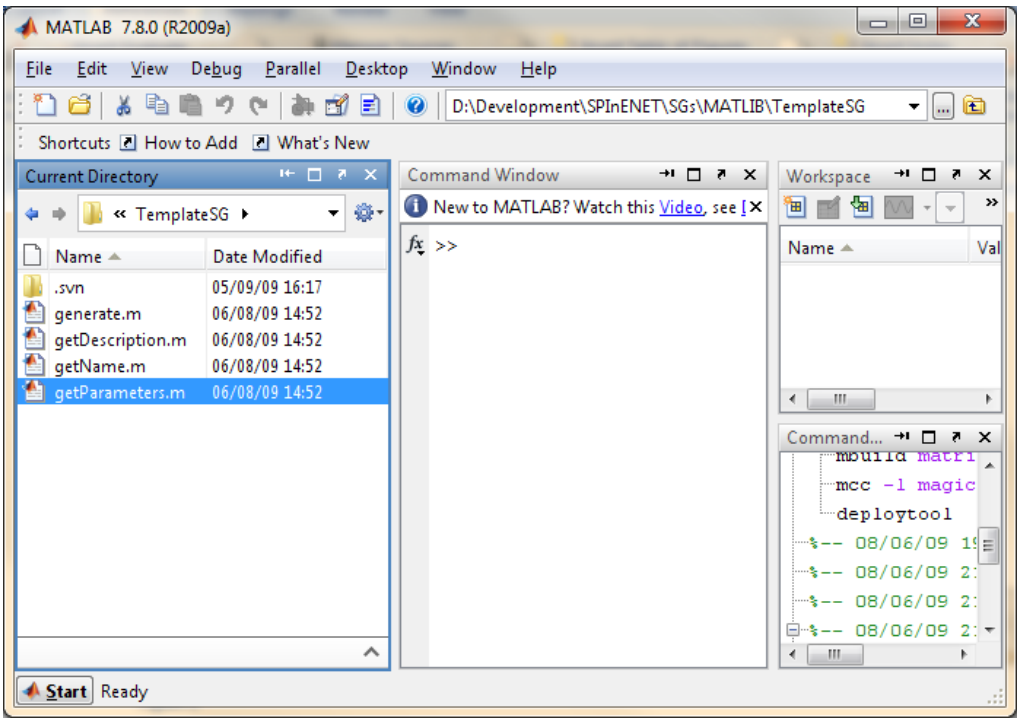
*generate*

```
% Template function for scenario generator
% Input parameter must be a varargin, the length of the cell
% array being passed will be consistent with what is specified
% in "getParameters"
% To extract the input parameters from the varargin cell
% array, the  notation varargin{index}, note the use of curly
% brackets to specify the index.

% The returned array should be a 3d array, whose dimensions
% are: [scenario, timeindex, otherindexingsets]
% The system defined parameter "TreeStructure" is used here,
% see documentation for further details. If for example the
% specified treestructure is [0 3 2 1], we have 4 scenarios,
% that branch at 3 different timeperiods. The total number of
% timeperiods is herby 4 (branching periods +1).
% Number of Scenarios and Number of timeperiods can be
% inferred from the treestructure as:
% NS=length(tree)   and   NT=max(tree)+1

function s=generate(varargin)

 % Check for the correct number of input arguments

if(length(varargin) ~= 6)

  s = 0;

  return;

end


%Extract parameters from packed input cell array

NI = varargin{1};

tree = varargin{2};

seed = varargin{3};


%Preallocates the matrix, size of s is [NS, NT,NI]

s = zeros(length(tree), max(tree)+1, NI);

%Fills it with random vectors of size scenarios, timeperiods

for x=1:NI

   s(:,:,x) = repmat(seed+x, length(tree), max(tree)+1);

end
```

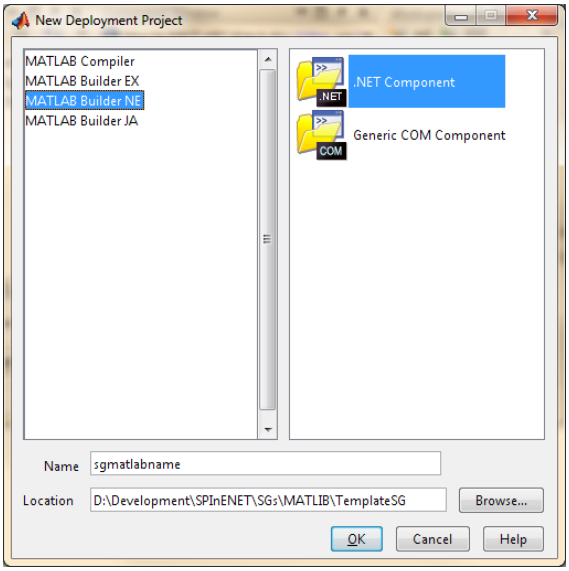
`

To make the developed scenario generator available to SPInE, the next step is to compile and pack the implemented m files into a library, using the .NET Builder toolbox. Step by step instructions are provided below:

Navigate to the template folder with MATLAB explorer:

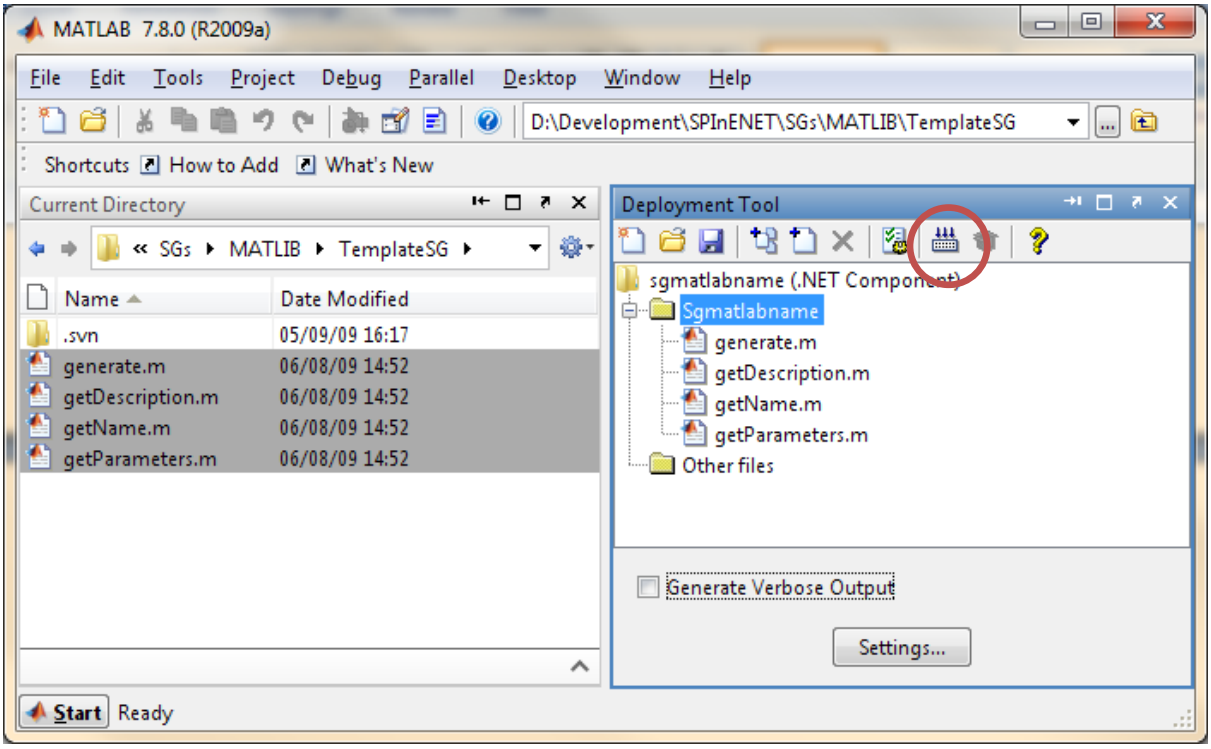



Create a new deployment project from MATLAB main menu, choosing MATLAB Builder NE as type of project and .NET Component as subproject. Name the project with the prefix “sgmatlab” as shown in the figure below:

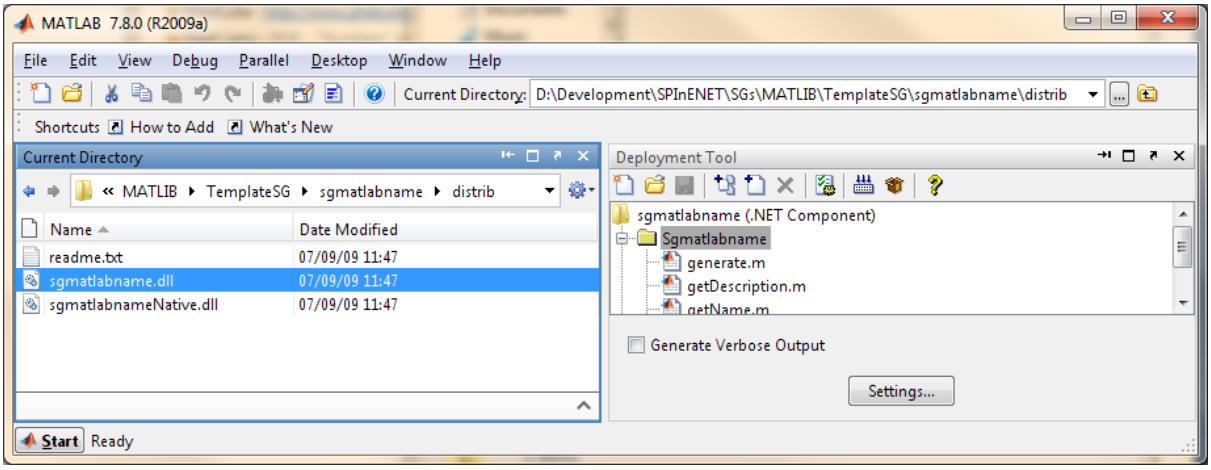


`

Drag and drop the functions which are part of the scenario generator into the deployment tool window, to make them part of the created library, as shown in the figure below, then click the icon Build (in the red circle in the figure).




After the compilation process has been completed, a new folder structure can be found in the project folder. In the subdirectory /distrib, the only file needed is the DLL that has the projectname as name, as shown in the figure below. That DKK must be copied in the folder specified in SPInE's option file as the scenario generators folder.



`

### *Modeller's perspective*

The use of MATLAB developed scenario generators follows the same syntax as the use of other SGs. The choice of the SG to be used and of its parameters is done in the declaration of the random parameter. As an example, to associate the SAMPL random parameter testSG, declared as:

```
scenarioset scenario;
set time;
set otherindex;
random param testSG{scenario, time, otherindex};
```

to the scenario generator TestMATLABNET defined in the previous part of this appendix, that wants from the user just the parameter seed of type `double` (see declaration of the function `getParameters`), the modeller should change the declaration line of `testSG` to the following:

```
random param testSG{scenario, time, otherindex} sg TestMATLAB-
NET(2.55);
```

where 2.55 is the seed for the random number generator inside the SG.

`

# Appendix B SAMPL syntax for random parameter declaration

Random parameter declarations have a list of optional attributes, optionally separated by commas, to accommodate the specification of the source scenario generator, the syntax has been extended to the following:

**random param** *name* $alias_{opt}$ $indexing_{opt}$ , $attributes_{opt}$ , $sgspecification_{opt}$

Where *attributes* may be any of the following:

*attribute:*

*binary*

*integer*

*symbolic*

*relop expr*

*in expr*

*= expr*

*default expr*

*relop:*

*< <= = == != <> > >=*

*sgspecification:*

*sg name (*$parameterlist_{opt}$*)*

*name:* a string identifying a scenario generator in the library

*parameterlist:* comma separated list of parameters, the interpretation of which is up to the external SG specified by *name*

`

# Appendix C ALM Model

## *Model formulation*

The asset/liability management model can be stated as follows: an investor faces the problem of creating a portfolio allocating a set of assets belonging to a universe *I*; each assets class is characterised by a price *P*. The goal of the investor is to maximise the portfolio wealth at the end of a predefined time horizon *T*. He needs to take into account future obligations (liabilities) *L*, and the fact that each trade has an associated transaction cost expressed by the fraction *g*. In each time period of the time horizon, and for each asset considered, the investor needs to decide the amount of assets to buy, to sell and to hold. Table 1 above shows a possible definition of the entities for such a model.

| Type | Name | Notation | Description | Range//Dimensions |
|---|---|---|---|---|
| **Indices** | *ASSETS* | I | Assets classes | i = 1 ... I; I=10 |
| **(sets)** | *TIME* | T | Time periods | t = 1 ... T; T=4 |
| **Parameters** | *price* | $P_{i\,t}$ | Price of asset i at time period t | ASSETS,TIME |
| **(data)** | *liabilities* | $L_t$ | Liability at time period t | TIME |
| | *Initialholdings* | $H_{i0}$ | Initial portfolio composition | ASSETS |
| | *income* | $F_t$ | Funding in time period t | TIME |
| | *Tcost* | G | Transaction cost as % of trade value | |
| **Variables** | *hold* | $H_{it}$ | Quantity of assets i to hold in time period t | ASSETS,TIME |
| | *sell* | $S_{it}$ | Quantity of assets i to sell in time period t | ASSETS,TIME |
| | *buy* | $B_{it}$ | Quantity of assets i to buy in time period t | ASSETS,TIME |

**Table 15 ALM model entities**

### *Asset holding constraints*

During the planning horizon the portfolio is re-balanced at discrete points in time (beginning of each time period). The model tends to buy the assets with the highest return expectation and sells the ones with poor performance. The *asset holding*

`

*constraint* enforces the evolution of the portfolio composition over time; it is expressed using two constraints, one that takes into consideration the initial holdings, one that doesn't. At time-periods t>1 the amount of each individual asset held in the portfolio is associated with the holding amount for each asset during the previous time-period.

$$H_{i1} = H_{i0} + B_{i1} - S_{i1} \quad , \; i=1\ldots I \qquad \textbf{6.1}$$

$$H_{it} = H_{it-1} + B_{it} - S_{it} \quad , \; t=2\ldots T, \; i=1\ldots I \qquad \textbf{6.2}$$

*Fund balance constraints*

Throughout the planning period cash inflows and cash outflows occur. The former is due to the assets selling or to profitable performance of the assets along with additional funding, which the investor's company might obtain. The latter is due to the company's payments and other liabilities which have to be fulfilled as well as to the purchase of assets and the transaction costs associated with their trading (buying and selling). In other words, this constraint reflects the evolution of the fund balance of the investor over time.

$$(1-G)\sum_{i=1}^{I} P_{it}S_{it} - L_t + F_t = (1+G)\sum_{i=1}^{I} P_{it}B_{it} \qquad \textbf{6.3}$$

Figure 46 below illustrates the concept of fund balance.

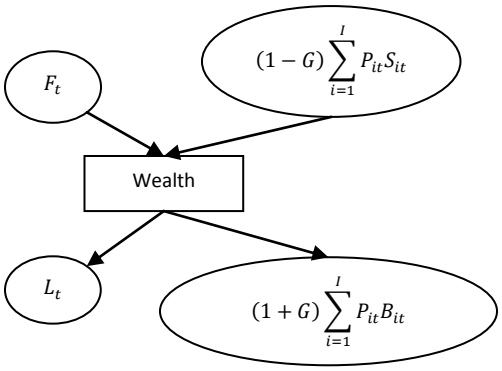



**Figure 46 Fund balance constraint**

Enforcing this constraint has the effect of linking outflows and inflows of cash, effectively creating a cash flow, where cash cannot be created nor destroyed.

`

*Objective function*

The goal of the investor is to maximise the terminal wealth of the portfolio. This can be expressed as in equation 6.4.

$$\sum_{i=1}^{I} P_{iT} H_{iT} \qquad \textbf{6.4}$$

The expression above can be used to calculate the market value of the portfolio for each time-period by substituting *T* with *t=1…T*.

## *ALM deterministic model in AMPL*

The AMPL code for the model, excluding the definition of the data tables is set out below. The comments in the code itself should provide enough guidance for a shallow understanding of the sections of the models, which follow the same order in which they have been presented. For a more detailed description of the features and the syntax of the language, the reader is referred to (Fourer et al., 2002).

```
#Parameters for indices
param NT:=4;
param NA:=23;
#Sets
set ASSETS := 1..NA;
set TIME :=1..NT;
#Parameters
param Tcost := 0.025;
param liabilities{TIME};
param initialholdings{ASSETS} := 0;
param income{TIME};
param price{TIME, ASSETS};
#Variables
var   hold{TIME, ASSETS} >=0;
var   buy{TIME, ASSETS} >=0;
var   sell{TIME, ASSETS} >=0;
```

`

```
#Objective function
maximize wealth: sum{a in ASSETS} price[NT,a]*hold[NT,a];
#Constraints
subject to
stockbalance1{a in ASSETS}:
      hold[1,a]=initialholdings[a]+buy[1,a]-sell[1,a];
stockbalance2{a in ASSETS,t in 2..NT}:
      hold[t,a]=hold[t-1,a]+buy[t,a]-sell[t,a];
fundbalance{t in TIME}:
      (1-Tcost)*(sum{a in ASSETS} price[t,a]*sell[t,a])
      - liabilities[t] + income[t] =
     (1+Tcost)*(sum{a in ASSETS} price[t,a]*buy[t,a]);
```

The data for the model (asset prices, incomes and liabilities) is specified in a separate file that is not reported here.

### *Stochastic ALM as deterministic equivalent in AMPL*

We introduce uncertainty in this model considering the future asset prices non deterministic; we therefore add another dimension to the model, which is called *scenario*, to include the realizations of the uncertain parameters in the form of scenario trees, as discussed in section 1.2.

Following the explicit non-anticipativity representation, some changes in the indexation of variables, parameters and constraints are needed, to take into consideration the fact that all decisions are now dependent on the scenario, and so is the parameter `price`. Moreover, the parameter `prob[SCENARIO]` is added to represent the probability of each scenario and the objective is now to minimize the expected final wealth. The changes are highlighted in bold in the model code below.

```
#Parameters for indices
param NT:=4;
param NA:=23;
param NS:=64;
#Sets
```

`

```
set ASSETS := 1..NA;
set TIME :=1..NT;
set SCENARIO := 1..NS;
#Parameters
param Tcost := 0.025;
param liabilities{TIME};
param initialholdings{ASSETS} := 0;
param income{TIME};
#Stochastic related parameters
param prob{SCENARIO} := 1/NS;
param price{TIME, ASSETS, SCENARIO};
#Variables
var   hold{TIME, ASSETS, SCENARIO} >=0;
var   buy{TIME, ASSETS, SCENARIO} >=0;
var   sell{TIME, ASSETS, SCENARIO} >=0;
#Objective function
maximize wealth: sum{s in SCENARIO} prob[s] * (sum{a in ASSETS}
price[NT,a,s]*hold[NT,a,s]);
#Constraints
subject to
stockbalance1{a in ASSETS, s in SCENARIO}:
      hold[1,a,s]=initialholdings[a]+buy[1,a,s]-sell[1,a,s];
stockbalance2{a in ASSETS,t in 2..NT, s in SCENARIO}:
      hold[t,a,s]=hold[t-1,a,s]+buy[t,a,s]-sell[t,a,s];
fundbalance{t in TIME, s in SCENARIO}:
      (1-Tcost)*(sum{a in ASSETS} price[t,a,s]*sell[t,a,s])
      - liabilities[t] + income[t] =
     (1+Tcost)*(sum{a in ASSETS} price[t,a,s]*buy[t,a,s]);
```

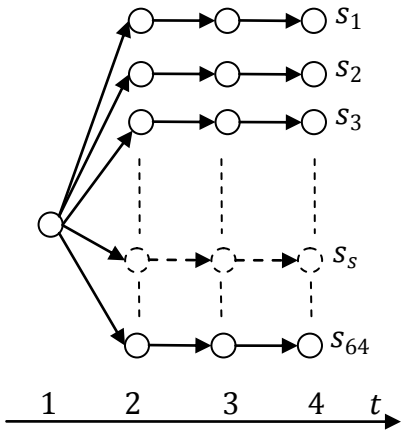
To complete the explicit non-anticipativity representation, a structure must be enforced to ensure that only the information available at each decision node influences the decision itself. We have therefore to add to the model the non-anticipativity constraints, which depend on the tree structure of our choice.

`

We consider two different tree shapes and, aside the graphical representation of the tree, the non-anticipativity constraint(s) for the variable hold are listed; to ensure a correct representation of the problem, similar constraints must be defined for all the decision variables.

Two stage tree, 64 scenarios

The model is two stage, with stage 1 including all decisions taken for t=1 and stage 2 including all the others. One non-anticipativity constraint is needed for each variable to ensure that, for t=1, the values of the variables remain constant for all scenarios.




```
# Stage1
na1{a in ASSETS, s in 2..NS}:
hold[1,a,1]=hold[1,a,s];
```

**Figure 47 Event tree for two-stage formulation**

Four stages tree, 64 scenarios, 4 branches at each stage

To ensure this kind of structure, in which the time index is equal to the stage number, one constraint template is needed for stage 1, four for stage 2 and sixteen for stage 3, for each variable.

`

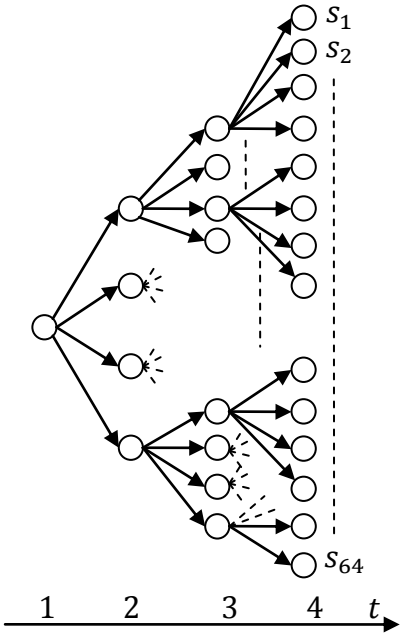



**Figure 48 Event tree for multi stage ALM model**

```
# Stage1

na1{a in ASSETS, s in 2..NS}:
hold[1,a,1]=hold[1,a,s];

#Stage 2

na1_s2_1{ a in ASSETS,s in 2..16}:
hold[2,a,s]=hold[2,a,1];

…

na1_s2_4{ a in ASSETS,s in 50..64}:
hold[2,a,s]=hold[2,a,49];

#Stage 3

na1_s3_1{a in ASSETS,s in 2..4}:
hold[3,a,s]= hold[3,a,1];

na1_s3_2{ a in ASSETS,s in 6..8}:
hold[3,a,s]= hold[3,a,5];

…

na1_s3_16{a in ASSETS,s in 62..64}:
hold[3,a,s]= hold[3,a,61];
```

## *Stochastic ALM formulated in SAMPL*

The SAMPL formulation of the model is reported below; besides the non-anticipativity constraints, which are not part of it, objective and constraints are identical to the DEQ formulation and therefore are not reported.

```
# Parameters for set ranges
param NT:=4; param NS:=64; param NA:=23;
# Sets
set ASSETS := 1..NA; set TIME :=1..NT;
# Stochastic information
scenarioset SCENARIO:=1..NS;
random param price{TIME, ASSETS, SCENARIO};
probability Prob{SCENARIO}:=1/NS;
#PARAMETERS : VECTORS (read from database!)
param liabilities{TIME}; param initialholdings{ASSETS} := 0;
param income{TIME}; param target{TIME}; param Tcost:=0.025;
```

`

```
#VARIABLES

var   hold{t in TIME,a in ASSETS, s in SCENARIO} >=0;

var   buy{t in TIME,a in ASSETS, s in SCENARIO} >=0;

var   sell{t in TIME,a in ASSETS, s in SCENARIO} >=0;
```

Now, depending on the desired event tree, the definition of the tree shape and the partition of decision variables into stages is, for the two-stage problem:

```
#Tree shape and staging for two stage ALM problem

tree thetree := twostage; #Specify a two-stage tree

let {t in TIME, a in ASSETS, s in SCENARIO} hold[t,a,s].stage := if
t=1 then 1 else 2;

let {t in TIME, a in ASSETS, s in SCENARIO} buy[t,a,s].stage := if
t=1 then 1 else 2;

let {t in TIME, a in ASSETS, s in SCENARIO} sell[t,a,s].stage := if
t=1 then 1 else 2;
```

or, for the multi stage problem:

```
#Tree shape and staging for multistage ALM problem

tree thetree := nway{4}; #Specify a tree with 4 branches at each
stage

let {t in TIME, a in ASSETS, s in SCENARIO} hold[t,a,s].stage := t;

let {t in TIME, a in ASSETS, s in SCENARIO} buy[t,a,s].stage := t;

let {t in TIME, a in ASSETS, s in SCENARIO} sell[t,a,s].stage := t;
```

The compactness of this formulation in respect to the DEQ one is noticeable; it should be noted that by generating the problem in this way, the system can automatically generate the Wait and See and the Expected Value problems, and calculate the VSS and EVPI.

### *CCP and ICCP formulations (as DEQs) in AMPL*

The incorporation of chance constraints and integrated chance constraints into this model allows the planned strategy to have some degree of underfunding, that is, at some point in time, the liquidity incomes don't match the liabilities; in our model this can be implemented allowing the fund balance to be negative.

A reformulation of the fund balance constraint of equation 6.3 is given in equation 6.5. The formulation has been furthermore refined with the introduction of scenar-

`

ios, to reflect the fact that we are now examining the stochastic version of the model.

$$(1-g)\sum_{i=1}^{I} P_{its}S_{its} - (1+g)\sum_{i=1}^{I} P_{its}B_{its} + F_t \geq L_t \qquad \forall\, t\ in\ 1,\dots,T, \forall s\ in\ 1,\dots,S \qquad \textbf{6.5}$$

To allow underfunding, one approach is to transform the constraint above to a chance constraint; this allows a violation to that constraint with a certain probability among all scenarios.

The first step is to define the underfunding for each scenario;

$$(1-g)\sum_{i=1}^{I} P_{its}S_{its} - (1+g)\sum_{i=1}^{I} P_{its}B_{its} + F_t + U_{ts} - O_{ts} \geq L_t$$
$$\forall\, t\ in\ 1,\dots,T, \forall s\ in\ 1,\dots,S$$
$$\text{where } U_{ts}, O_{ts} \geq 0 \ \ \forall\, t\ in\ 1,\dots,T, \forall s\ in\ 1,\dots,S \qquad \textbf{6.6}$$

and finally

$$\overline{U}_s = \sum_{t=1}^{T} U_{ts}$$

where $U_{ts}$ and $O_{ts}$ are variables defined for all time periods and all scenarios.

Allowing underfunding in this model has a side effect: the investor can re-invest the amount of money that "appears" from the underfunding. This is not coherent with proper cash balancing; therefore another constraint must be added, to bind the investor to invest just the cash coming from the liquidation of assets and the income at that time period. We call this the *cash balance* constraint, and the formulation is as follows:

$$(1-g)\sum_{i=1}^{I} P_{its}S_{its} + F_t \geq (1+g)\sum_{i=1}^{I} P_{its}B_{its} \qquad \textbf{6.7}$$
$$\forall\, t\ in\ 1,\dots,T, \forall s\ in\ 1,\dots,S$$

The chance constraint can be written as:

$$P_{s\ in\ 1,\dots,S}\{\overline{U}_s \leq 0\} \geq R \qquad \textbf{6.8}$$

`

where *R* is a reliability level, that is the probability with which we want to satisfy the constraint. As seen in section 1.3, there is a deterministic equivalent formulation (see the system of equations 1.43) for CCP problem. The changes to the deterministic equivalent AMPL formulation follows:

```
param M := 50000;
param Reliability:=0.8; #do not underfund with probability 80%
var count{SCENARIO} binary;
var over{TIME, SCENARIO} >= 0;
var under{TIME,SCENARIO} >= 0;
var underDeviation{SCENARIO} >=0;
fundbalance{t in TIME, s in SCENARIO}:
  (1-Tcost)*(sum{a in ASSETS} price[t,a,s]*sell[t,a,s])
  - (1+Tcost)*(sum{a in ASSETS} price[t,a,s]*buy[t,a,s])
  + income[t] - over[t,s] + under[t,s] = liabilities[t];
cashbalance{t in TIME, s in SCENARIO}:
 (1-Tcost)*(sum{a in ASSETS} price[t,a,s]*sell[t,a,s]) + income[t]>=
 (1+Tcost)*(sum{a in ASSETS} price[t,a,s]*buy[t,a,s]);
underDevDef{s in SCENARIO}: sum{t in TIME} under[t,s] = underDevia-
tion[s];
CC{s in SCENARIO}: underDeviation[s] <= M * count[s];
cardCC: sum{s in SCENARIO} prob[s]*count[s] <= 1-Reliability;
```

The artifices introduced in the model due to the DEQ formulation are highlighted in bold. It is worth noticing that this formulation introduces one binary variable for each scenario.

This model easily spots one weakness of the chance constraints problems, which is the fact that they represent a qualitative risk measure. The scenarios that are allowed to underfund in the problem above, do indeed underfund, and they can do so by up to M; one scenario with very little underfunding is considered equally to one which underfunds by the maximum allowed. This does not take into consideration that the *amount* of underfunding has an important role too. For this reason, in

`

this case the integrated chance constraint approach might be preferable; the ICC takes the amount of underfunding into consideration, limiting the expected underfunding.

The formulation is:

$$E_{s\ in\ 1,\dots,S}[\overline{U}_s] \leq G \qquad \textbf{6.9}$$

which can be implemented in AMPL adding:

```
param G := 50000;
ICCP: sum{s in SCENARIO} prob[s]*underDeviation[s] <= G;
```

The additional variables mentioned in section 1.3 have not been added; normally the deterministic equivalent formulation of an integrated chance constraint requires the creation of an additional variable for each scenario.

### *CCP and ICCP formulations in SAMPL*

The formulation of the chance constraint using SAMPL extended syntax is:

```
CC: {probability s in SCENARIO: underDeviation[s] > 0} <= Reliability;
```

Similarly the integrated change constraint reads:

```
ICCP: expectation{s in SCENARIO} {underDeviation[s]} <= G;
```

It can be easily seen that the formulation using the extended syntax is much more compact and readable. Moreover, this reformulation allows the modelling system to use a solver that is especially designed to solve CCPs or ICCPs through specialized algorithms (see (Haneveld & van der Vlerk, 2006) for an example), whenever such a solver is available.

### *Robust formulations in AMPL*

In case more precise assumptions about the distribution of the random parameters cannot be made, reformulating the model as a robust optimization problem can help maintaining feasibility of the solution in the face of an uncertain future. Only a few and light assumptions in respect of the random parameters are made in the model of uncertainty U presented in section 1.2. We therefore model the future

`

assets prices as $\tilde{P}_{it}$ with values in $[P_{it} - \check{P}_{it}, P_{it} + \check{P}_{it}]$, where $P_{it}$ and $\check{P}_{it}$ are respectively the mean value and the half extension of the uniform distribution of asset *i* at time *t* and. It has to be noted that the prices are the only non-deterministic parameters of this model, and that they appear as elements in the matrix with a multiplier, as coefficients of the variables *S* and *B*:

$$(1-g)\sum_{i=1}^{I} P_{it}S_{it} - (1+g)\sum_{i=1}^{I} P_{it}B_{it} + F_t \geq L_t \qquad \textbf{6.10}$$

To avoid being too prolix on a topic – the formulation of robust optimisation problems – which is not central to this thesis, only the formulation given by Soyster (see section 1.2) is explicitly given here. The linear program that can be inferred from Soyster's formulation, as in 1.31, is reported below:

$$Z = \max cx$$

$$\begin{gathered} subject\ to \sum_{j} a_{ij}x_j + \sum_{j\in J_i} \check{a}_{ij}y_j \leq b_i \quad \forall\, i \\ -y_j \leq x_j \leq y_j \quad \forall\, j \\ l \leq x \leq u \\ y \geq 0. \end{gathered} \qquad \textbf{6.11}$$

The formulation requires the knowledge of the sets $J_i$ of coefficients in each row *i* that are subject to uncertainty, because an artificial variable $y_j$ must be created for each one of them. In the algebraic perspective, the procedure translates into recognizing the parameters that are defined as part of the model U (in this model, the prices) and add an artificial variable for each time they appear in each constraint. In our case, the only constraint involved is shown in equation 6.6, and the random parameter $\tilde{P}_{it}$ appears twice in it. We therefore proceed by creating two artificial variables $y_{Bit}$ and $y_{Sit}$ for each constraint, which obtains the final form:

$$\begin{gathered} (1+g)\sum_{i=1}^{I} P_{it}B_{it} - (1-g)\sum_{i=1}^{I} P_{it}S_{it} + (1+g)\sum_{i=1}^{I} \check{P}_{it}y_{Bit} \\ -(1-g)\sum_{i=1}^{I} \check{P}_{it}y_{Sit} \leq F_t - L_t \quad \forall\, t \in T \end{gathered} \qquad \textbf{6.12}$$

`

To complete the formulation, the constraints which link the artificial variables and the natural one have to be added, together with the bounds on the variables, namely:

$$\begin{aligned} -y_{Bit} \leq B_{it} \leq y_{Bit} \ , \ \forall\, t \in T, i \in A \\ y_{Sit} \leq S_{it} \leq y_{Sit} \ , \forall\, t \in T, i \in A \\ B_{it} \geq 0 \ and \ S_{it} \geq 0 \ , \forall\, t \in T, i \in A \end{aligned} \qquad \textbf{6.13}$$

Expressed in AMPL, the steps above are:

1) Declare the artificial variables and the parameters of the uniform distribution (as mean value $P_{it}$ I used the expected value of the realizations utilized for the SP problem, so just the additional parameter `amplitude` was needed)

```
param amplitude{TIME, ASSETS} := 10;
var artificialBuy{TIME, ASSETS} >= 0;
var artificialSell{TIME, ASSETS} >= 0;
```

2) Reformulate the fundbalance constraint to implement 6.12:

```
fundbalance{t in TIME}:
   (1+Tcost)*(sum{a in ASSETS} price[t,a]*buy[t,a]) -
   (1-Tcost)*(sum{a in ASSETS} price[t,a]*sell[t,a])+
     sum{a in ASSETS} (1+Tcost)*artificialBuy[t,a]*amplitude[t,a] -
     sum{a in ASSETS} (1-Tcost)*artificialSell[t,a]*amplitude[t,a]
       <= income[t] - liabilities[t];
```

3) Implement the other constraints:

```
robustBuy{t in TIME, a in ASSETS}: -artificialBuy[t,a] <= buy[t,a];
robustBuy2{t in TIME, a in ASSETS}: buy[t,a] <= artificialBuy[t,a];
robustSell{t in TIME,a in ASSETS}:-artificialSell[t,a] <= sell[t,a];
robustSell2{t in TIME, a in ASSETS}: sell[t,a]<=artificialSell[t,a];
```

It is now apparent that, even for this simple problem, the reformulation as deterministic equivalent takes the focus of the modeller away from the problem itself, to

`

concentrate with the definition of artificial variables and the reformulation of constraints.

### *Robust formulations in SAMPL*

Expressed using SAMPL extended syntax, the steps above are simplified. The definition of the random parameter is changed to:

```
random param randomPrice{t in TIME, a in ASSETS}
dist symmetric(price[t,a] - amplitude[t,a], price[t,a] + amplitude[t,a]);
```

This formal definition of the price gives all the needed information the modelling system regarding the uncertainty model U. The next step is to choose the form of the robust formulation, which is obtained via the following statement:

```
option RobustForm Soyster;
```

Finally, the constraint are expressed identically to the deterministic version, as:

```
fundbalance{t in TIME}:
      (1-Tcost)*(sum{a in ASSETS} randomPrice[t,a]*sell[t,a])
      - liabilities[t] + income[t] =
     (1+Tcost)*(sum{a in ASSETS} randomPrice [t,a]*buy[t,a]);
```

The system takes care of generating the artificial variables and the additional constraints automatically, thus allowing the modeller to concentrate on the problem instead of the formal specification of the uncertainty set.

To obtain the other formulations (Ben Tal and Nemirovsky, or Bertsimas and Sim), the modeller simply uses a different value for the RobustForm option. These two formulations require additional parameters to specify the desired trade-off between optimality and robustness. This parameter is specified in the constraint declaration, as:

```
fundbalance{t in TIME} suffix robustness gamma[t]:
      (1-Tcost)*(sum{a in ASSETS} randomPrice[t,a]*sell[t,a])
      - liabilities[t] + income[t] =
     (1+Tcost)*(sum{a in ASSETS} randomPrice [t,a]*buy[t,a]);
```

where gamma is an AMPL parameter containing the chosen robustness value.