Stochastic Constraint Programming: A Scenario-Based Approach

S. Armagan Tarim,
Cork Constraint Computation Centre, University College Cork, Ireland
Suresh Manandhar,
Department of Computer Science, University of York, United Kingdom
Toby Walsh,
National ICT Australia and School of Computer Science and Engineering,
University of New South Wales, Australia

Abstract

To model combinatorial decision problems involving uncertainty and probability, we introduce scenario based stochastic constraint programming. Stochastic constraint programs contain both decision variables, which we can set, and stochastic variables, which follow a discrete probability distribution. We provide a semantics for stochastic constraint programs based on scenario trees. Using this semantics, we can compile stochastic constraint programs down into conventional (non-stochastic) constraint programs. This allows us to exploit the full power of existing constraint solvers. We have implemented this framework for decision making under uncertainty in stochastic OPL, a language which is based on the OPL constraint modelling language [Hentenryck et al., 1999]. To illustrate the potential of this framework, we model a wide range of problems in areas as diverse as portfolio diversification, agricultural planning and production/inventory management.

Keywords: constraint programming, constraint satisfaction, reasoning under uncertainty

Content areas: constraint programming, constraint satisfaction, reasoning under uncertainty

1 Introduction

Many decision problems contain uncertainty. Data about events in the past may not be known exactly due to errors in measuring or difficulties in sampling, whilst data about events in the future may simply not be known with certainty. For example, when scheduling power stations, we need to cope with uncertainty in future energy demands. As a second example, nurse rostering in an accident and emergency department requires us to anticipate variability in workload. As a final example, when constructing a balanced bond portfolio, we must deal with uncertainty in the future price of bonds. To deal with such situations, [27] proposed an extension of constraint programming, called *stochastic constraint programming*, in which we distinguish between decision variables, which we are free to set, and stochastic (or observed) variables, which follow some probability distribution. A semantics for stochastic constraint programs based on policies was proposed and backtracking and forward checking algorithms to solve such stochastic constraint programs were presented.

In this paper, we extend this framework to make it more useful practically. In particular, we permit multiple chance constraints and a range of different objectives. As each such extension requires large changes to the backtracking and forward checking algorithms, we propose instead a scenario based view of stochastic constraint programs. One of the major advantages of this approach is that stochastic constraint programs can then be compiled down into conventional (non-stochastic) constraint programs. This compilation allows us to use existing constraint solvers without any modification, as well as call upon the power of hybrid solvers which combine constraint solving and integer programming techniques. We also propose a number of techniques to reduce the number of scenarios and to generate robust solutions. This framework combines together some of the best features of traditional constraint satisfaction, stochastic integer programming [24], and stochastic satisfiability [14]. We have implemented this framework for decision making under uncertainty in a language called Stochastic OPL. This is an extension of the OPL constraint modelling language [10]. Finally, we describe a wide range of problems that we have modelled in Stochastic OPL that illustrate some of its potential.

2 Motivation Example

We consider a stochastic version of the "template design" problem. The *deterministic* template design problem (prob002 in CSPLib, http://www.csplib.org) is described as follows. We are given a set of variations of a design, with a common shape and size and such that the number of required "pressings" of each variation is known. The problem is to design a set of templates, with a common capacity to which each must be filled, by assigning one or more instances of a variation to each template. A design should be chosen that minimises the total number of "runs" of the templates required to satisfy the number of pressings required for each variation. As an example, the variations might be for cartons for different flavours of cat food, such as fish or chicken, where ten thousand fish cartons and twenty thousand chicken cartons need to be printed. The problem would then be to design a set of templates by assigning a number of fish and/or chicken designs to each template such that a minimal number of runs of the templates is required to print all thirty thousand cartons. Proll and Smith address this problem by fixing the number of templates and minimising the total number of pressings [20].

In the *stochastic* version of the problem, the demand for each variation is uncertain. We adopt the Proll–Smith model in what follows, extending it to comply with the stochastic demand assumption. We use the following notation for problem parameters: N, number of variations; T, number of templates; S, number of slots on each template; c_h, unit scrap cost for excess inventory; c_p, unit shortage cost. The decision variables are: $a_{i,j}$, number of slots designated to variation i, on template j; R_j, number of required "runs" of template j. For convenience, we define the following auxiliary variables: x_i, total production for variation i; e_i, total scrap in variation i; b_i, total shortage in variation i. There are also stochastic variables d_i representing stochastic demand for variation i.

This problem can be modelled as stochastic constraint optimization problem. There is a constraint to ensure that the total number of slots designated to variations is exactly the number of slots available, which is S.

$$\sum_{i=1}^{N} a_{ij} = S, \quad \forall j \in \{1, ..., T\}, \tag{1}$$

There is also a constraint to determine the total production in each variation.

$$\sum_{j=1}^{T} a_{ij} R_j = x_i, \quad \forall i \in \{1, ..., N\}. \tag{2}$$

And there are two constraints to determine the amount of shortage and scrap for each variation.

$$e_i = \max\{0, x_i - d_i\}, \forall i \in \{1, ..., N\} \tag{3}$$

$$b_i = -\min\{0, x_i - d_i\}, \forall i \in \{1, ..., N\}. \tag{4}$$

Our objective is to minimise the total expected shortage and scrap costs,

$$\min E\left(\sum_{i=1}^{N} c_p b_i + c_h e_i\right) \tag{5}$$

where $E(.)$ denotes the expectation operator. From Eqs. (3) and (4) it is clear that b_i and e_i are random variables, since their values depend on the realization of random demand.

Demand uncertainty necessitates carrying buffer-stocks. Overstock leads to high inventory holding and scrap costs. On the other hand, insufficient buffer stocks are also financially damaging, leading to stock-outs and loss of customer satisfaction. An alternative method to deal with such uncertainty is to introduce service-level constraints instead of using shortage costs. In this case a service-level constraint is expressed in the form of a chance constraint as follows,

$$\Pr\{x_i - d_i \geq 0\} \geq \alpha_i, \quad i \in \{1, ..., N\} \tag{6}$$

where $\Pr(.)$ represents the probability function and α denotes a target service-level.

It may also be natural to consider measures such as the worst case performance, other moments of expected performance like variance which is a proxy for risk, the probability of attaining a predetermined performance goal, and even, in certain types of problems like engineering design problem, we may want to minimize the spread (i.e. minimize the difference between the least and the largest value of the objective function). Note that stochastic variables need not be independent (as assumed in [27]). For example, if demand for a certain item is low in the first quarter, it is more likely to be low in the second quarter as well.

Unfortunately, each of these extensions requires a major modification to the backtracking and forward checking algorithms presented in [27]. We therefore take a different track which permits us to define these extensions without major modifications to the solution methods. We define a new and equivalent semantics for stochastic constraint programs based on scenarios which permits the above extensions, namely conditional probabilities, multiple chance constraints, as well as a much wider range of goals. This scenario-based view permits stochastic constraint programs to be compiled down into regular (non-stochastic) constraint programs. We can therefore use traditional constraint satisfaction and optimization algorithms, as well as hybrid methods that use techniques like integer linear programming.

3 Scenario-based semantics

A stochastic constraint satisfaction problem consists of a 6-tuple $\langle V, S, D, P, C, \theta \rangle$. V is a set of decision variables, and S is a set of stochastic variables. D is a function mapping each element of V and each element of S to a domain of potential values. A decision variable in V is *assigned* a value from its domain. P is a function mapping each element of S to a probability distribution for its associated domain. C is a set of constraints, where a constraint $c \in C$ on variables $x_i, \ldots, x_j$ specifies a subset of the Cartesian product $D(x_i) \times \ldots \times D(x_j)$ indicating mutually-compatible variable assignments. The subset of C that constrain at least one variable in S are *chance constraints*, h. θ_h is a threshold probability in the interval $[0, 1]$, indicating the fraction of scenarios in which the chance constraint h must be satisfied. Note that a chance constraint with a threshold of 1 is equivalent to a hard constraint.

A stochastic CSP consists of a number of *decision stages*. In a one-stage stochastic CSP, the decision variables are set before the stochastic variables. In an m-stage stochastic CSP, V and S are partitioned into n disjoint sets, $V_1, \ldots, V_m$ and $S_1, \ldots, S_m$. To solve an m-stage stochastic CSP an assignment to the variables in V_1 must be found such that, given random values for S_1, assignments can be found for V_2 such that, given random values for S_2, ..., assignments can be found for V_m so that, given random values for S_m, the hard constraints are satisfied and the chance constraints are satisfied in the specified fraction of all possible scenarios.

In the policy based view of stochastic constraint programs of [27], the semantics is based on a tree of decisions. Each path in a policy represents a different possible scenario (set of values for the stochastic variables), and the values assigned to decision variables in this scenario. To find satisfying policies, backtracking and forward checking algorithms, which explores the implicit AND/OR graph, are presented. Stochastic variables give AND nodes as we must find a policy that satisfies all their values, whilst decision variables give OR nodes as we only need find one satisfying value. An alternative semantics for stochastic constraint programs, which suggests an alternative solution method, comes from a scenario-based view [2].

In the scenario-based approach, a scenario tree is generated which incorporates all possible realisations of discrete random variables into the model explicitly. A tree representation of a 3-stage problem, with 2 possible states at each stage, is given in Fig.1.

Scenarios deal with uncertain aspects (e.g. the economic conditions, the state of the financial markets, the level of demand) of the operating environment relevant to the problem. Hence, the future uncertainty is described by a set of alternative scenarios. The number of scenarios as well as the progression of the scenarios from one period to another is problem specific. A path from the root to an extremity of the event tree represents a *scenario*, $\omega \in \Omega$, where Ω is the set of all possible scenarios. With each scenario a given probability is associated. If S_i is the ith random variable on a path from the root to the leaf representing scenario ω, and a_i is the value given to S_i on the ith stage of this scenario, then the probability of this scenario is again $\prod_i \Pr(S_i = a_i)$.

Thus, a scenario is associated with each path in the policy. Within each scenario, we have a conventional (non-stochastic) constraint program to solve. We simply replace the stochastic variables by the values taken in the scenario, and ensure that the values found for the decision variables are consistent across scenarios as certain decision variables are shared across scenarios. For instance, node 1 of the tree in Figure 1 corresponds to the first stage and associated decisions are identical for all

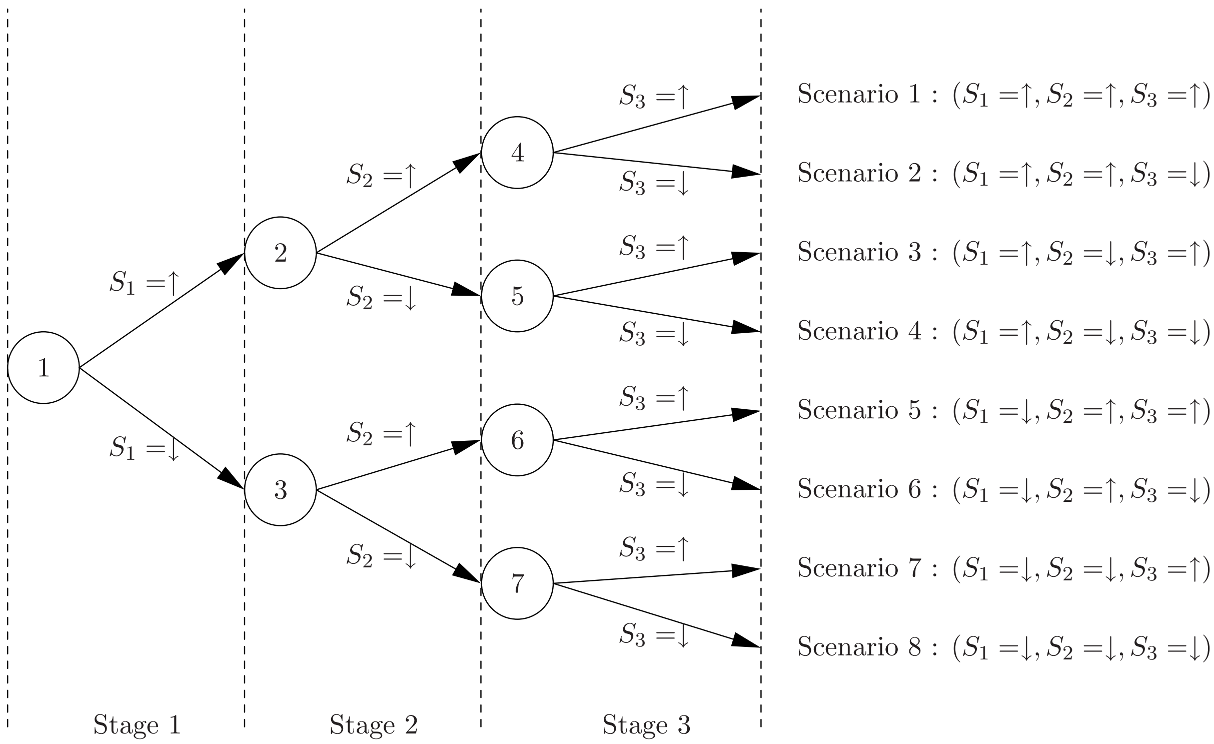

Figure 1: A tree representation of a Stochastic CP

scenarios. Note that in stage 2, the decisions of scenarios 1 to 4 are identical. Similarly in stage 2, the decisions of scenarios 5 to 8 are identical.

Constraints are defined (as in traditional constraint satisfaction) by relations of allowed tuples of values, and can be implemented with specialized and efficient algorithms for consistency checking. The great advantage of this approach is that we can use conventional constraint solvers to solve stochastic constraint programs. We do not need to implement specialized solvers. The scenario-based view of stochastic constraint programs also allows later stage stochastic variables to take values which are conditioned by the earlier stage stochastic variables. This is a direct consequence of employing the scenario representation, in which stochastic variables are replaced with their scenario dependent values.

Of course, there is a price to pay as the number of scenarios grows exponentially with the number of stages. However, our results show that a scenario-based approach is feasible for many problems. Indeed, we observe much better performance using scenario-based approach on the book production planning example of Walsh [27] compared to the tree search methods. In addition, as we discuss later, we have developed a number of techniques like Latin hypercube sampling to reduce the number of scenarios considered.

The results in Table 1 on the book production planning example from [27] show that the scenario-based approach offers much better performance on this problem than the forward checking or backtracking tree search algorithms. Failures and choice points denote the number of failures encountered during the resolution and the number of choices needed to produce the solution, respectively.

Constraint satisfaction is NP-complete in general. Not surprisingly, stochastic constraint satisfaction moves us up the complexity hierarchy.

	Backtracking (BT)		Forward Checking (FC)		Scenario-Based (SB)		
No. Stages	Nodes	CPU/sec	Nodes	CPU/sec	Failure	Choice Points	CPU/sec
1	28	0.01	10	0.01	4	5	0.00
2	650	0.09	148	0.03	4	8	0.02
3	17,190	2.72	3,604	0.76	8	24	0.16
4	510,356	83.81	95,570	19.07	42	125	1.53
5	15,994,856	3,245.99	2,616,858	509.95	218	690	18.52
6	–	–	–	–	1260	4035	474.47

Table 1: A Comparison of Policy Search Methods and Scenario Tree Approach

Theorem 1. *Stochastic constraint satisfaction problems are PSPACE-complete.*

Proof: Membership in PSPACE follows from the existence of a naive depth-first and/or search tree algorithm for solving stochastic constraint satisfaction problems. This algorithm recurses through the variables in order, making an "and" branch for a stochastic variable and an "or" branch for a decision variable. The algorithm requires linear space in the number of variables.

To show completeness, we reduce the satisfiability of quantified Boolean formula to a stochastic constraint satisfaction problem. Each existential Boolean variable in the quantified Boolean formula is mapped to a Boolean decision variable in the stochastic CSP. Each universal Boolean variable in the quantified Boolean formula is mapped to a Boolean stochastic variable in the stochastic CSP. Such variables have equal probability of being $true$ or $false$. Each clause is replaced by the equivalent constraint which is to be satisfied in all possible scenarios. The reduction is linear in the size of the original quantified Boolean formula. The quantified Boolean formula is satisfiable iff the stochastic CSP is itself satisfiable. $\diamond$

4 Stochastic OPL

We have implemented this framework on top of the OPL constraint modelling language [10]. An OPL model consists of two parts: a set of declarations, followed by an instruction. Declarations define the data types, constants and decision variables. An OPL instruction is either to satisfy a set of constraints or to maximize/minimize an objective function subject to a set of constraints. We have extended the declarations to include the declaration of stochastic variables, and the instructions to include chance constraints, and a range of new goals like maximizing the expectation of an objective function.

4.1 Constant and Variable declarations

Stochastic variables are set according to a probability distribution using a command of the form:

```
stoch <Type> <Id> <Dist>;
```

Where `<Type>` is (as with decision variables) a data type (e.g. a range of values, or an enumerated list of values), `<Id>` is (as with decision variables) the variable name, and `<Dist>` defines the probability distribution of the stochastic variable(s). Probability distributions include `uniform`, `poisson(lambda)`, and user defined via a list of (not necessarily normalized) values. Other types of distribution can be supported as needed. We insist that stochastic variables are arrays, with the last index describing the stage. Here are two different data representations of stochastic variables:

```
stoch float market[Years] =
    [<0.05 (0.34), 0.07 (0.66)>,<0.02 (0.25), 0.04 (0.25), 0.09 (0.5)>];
stoch int demand[Period] =
    [2 (0.25), 3 (0.75), 4 (0.35), 5(0.15), 7 (0.50), 8 (0.40), 9 (0.60)];
```

In the first, we have a float variable in the first year which is either 0.05 (with probability 0.34) or 0.07 (with probability 0.66). The notation, "$< . >$", is convenient for problems in which random variables are independent. In the second, we have the stochastic variable `demand`, which takes the value of either 2 or 3 in the first period. The value of the random variable in the second period depends on the first period's realization. It is $\{4, 5, 7\}$ if the first period's demand is 2, and $\{8, 9\}$ if it is 3. This notation is convenient especially for problems involving conditional probabilities.

The constants are declared as in OPL with the exception of the case where their values depend on the stochastic variables. In a financial planning problem, if it is assumed that the financial instrument return rates solely depend on unpredictable market then the return matrix `return[Instr,Period]` must be related to the stochastic variable `market[Period]`. The dependence of `return` on `market` is denoted by joining them together by a hat, ^, as in

```
return[Instr,Period]^market = ...;
```

4.2 Constraint posting

We can post both hard constraints (as in OPL) and chance constraints. Chance constraints hold in some but not necessarily all scenarios. They are posted using a command of the form:

```
prob(<Constraint>) <ArithOp> <Expr>;
```

Where `<Constraint>` is any OPL constraint, `<ArithOp>` is any of the arithmetically comparison operations (=,<>,<,>, <=, or >=) and `<Expr>` is any arithmetic expression (it may contain decision variables or may just be a rational or a float in the range 0 to 1). For example, the following command specifies the chance constraint that in each quarter the demand (a stochastic variable) does not exceed the production (a decision variable) plus the stock carried forward in each quarter (this auxiliary is modelled, as in conventional constraint programming, by a decision variable) with 80% probability:

```
forall(i in 1..n)
   prob(demand[i] <= production[i]+stock[i]) >= 0.80;
```

Constraints which are not chance constraints are hard and have to hold in all possible scenarios. For example, the stock carried forwards is computed via the hard constraint:

```
forall(i in 1..n)
   stock[i+1] = max(0,stock[i] + production[i] - demand[i]);
```

4.3 Optimization

Stochastic OPL supports both stochastic constraint satisfaction and optimization problems. We can maximize or minimize the expectation of an objective function. As an example of an expected value function, we'll consider the stochastic template design problem of Sec.2, in which the expected total cost of scrap and shortage is minimised. This can be specified by the following (partial) model:

```
minimize expected(cost)
subject to
  cost = sum(i in 1..n) ch*max(0,x[i]-d[i])-cp*min(0,x[i]-d[i]);
  ...
```

We can also model risk. For example, we may wish to reason about the mean and variance in the return for a portfolio selection problem [16]. Markowitz's mean/variance model provides a framework to examine the tradeoff between the expected value and its variability. In the mean/variance model, the expected value plus a constant ($\lambda \leq 0$) times the standard deviation –standard deviation is used as a surrogate for risk– is maximized. However, since the risk expression of Markowitz is very complex to reason with, we consider the simplification introduced in [11] and [12].

In [12], the absolute deviation function, K, is introduced as $K = |Q - E\{Q\}|$, where the random variable Q denotes the objective function value. [11] demonstrates that mean absolute deviation function can remove most of the difficulties associated with the standard deviation function. Stochastic OPL supports Markowitz's mean/variance model, where the surrogate risk measure is the absolute deviation function, with the command:

```
maximize mv(<Expr>,λ)
```

Numerical experiments in portfolio optimization show that the Konno-Yamazaki risk model generates results quite similar to that of Markowitz's quadratic mean/variance model. The mean absolute deviation risk model is discussed in Sec.9.

Stochastic OPL also supports a number of other optimization goals. For example:

```
minimize spread(profit)
maximize downside(profit)
minimize upside(cost)
```

The spread is the difference between the value of the objective function in the best and worst scenarios, whilst the downside (upside) is the minimum (maximum) objective function value a possible scenario may take.

5 Compilation of Stochastic OPL

These stochastic extensions are compiled down into conventional (non-stochastic) OPL models automatically by exploiting the scenario-based semantics. The compiler is written in Lex and Yacc, with a graphical interface in Visual C++. A demonstration version of the stochastic OPL compiler and example problems can be downloaded from http://www-users.cs.york.ac.uk/∼at/project. In what follows, the compilation of decision variables, constants, hard constraints, chance constraints and objective functions are discussed.

5.1 Compiling Constants, Variables and Hard Constraints

Compilation involves replacing stochastic variables by their possible values, and decision variables by a ragged array of decision variables, one for each possible scenario. Constants that depend on stochastic variables also require ragged arrays. We identify positions in the scenario tree by the stage (e.g. first stage, or second stage) and by an ordering over the states at this stage. We therefore declare:

```
struct AStruct {
        AStageRange stage;
        int state; };
```

where `AStageRange` is a stage index range and is extracted from the stochastic variable declaration. By means of this structure, the relevant <stage, state> pairs are declared:

```
{AStruct}
Nodes = { <stage,state> | stage in AStageRange & state in 1..nbNodes[stage] };
```

where `nbNodes[stage]` array denotes the number of states at the beginning of each stage and is extracted from probability data.

To build the certainty equivalent model using the notion of scenarios, a matrix `ScenTree` is declared and a reference to each node is made via <`stage,ScenTree[stage,scen]`> where `scen` denotes a scenario. Variable and constant compilations are performed by means of <`stage,ScenTree[stage,scen]`> notation and the following rules:

- *Constants with No-Stochastic Dependence*:
  Constants declared as independent of stochastic variable, are not altered.

- *Constants with Stochastic Dependence*:
  In the compiled model, in each stage for each random realization, there should be a unique value of each constant with stochastic dependence. This is achieved by replacing constants `C[...,t,...]^R` with `C[...,<t,ScenTree[t+1,scen]>,...]` where `t` stands for the stage index.

- *Variables with No-Stage Index* :
  For each scenario there should be a unique decision variable. Therefore, a decision variable without a stage index, `X[`ind_1`,...,`ind_n`]`, is replaced with `X[`ind_1`,...,`ind_n`,scen]`.

- *Variables with Stage Index* :
  Stage indexed variables are modified the same way as constants that depend on the stochastic variable: `X[...,<t,ScenTree[t,scen]>,...]` replaces `X[...,t,...]`.

Once the variables and constants are transformed and the range of possible scenarios, `Scenarios`, is determined then the compilation of stochastic hard constraints into equivalent deterministic ones requires only a `forall` statement to cover all possible scenarios: `forall(scen in Scenarios)` { *all constraints* };

The following financial planning example (see section 8.1 for the problem description), with a stage range `1..N`, demonstrates the application of compilation rules. A mathematical formulation of the problem, and corresponding stochastic and certainty-equivalent OPL representations thereof are given in the appendix. As explained above, all problem constraints must be given as a part of the `forall(scen in Scenarios) { <Constraints> };` statement.

- `capital = wealth[1];` *// "capital" is a constant, "wealth" is a decision variable*
  is compiled into `capital = wealth[<1,ScenTree[1,scen]>];`

- `finalwealth = wealth[N+1];` *// "finalwealth" is a decision variable*
  is compiled into `finalwealth[scen] = wealth[<N+1,ScenTree[N+1,scen]>];`

- `wealth[p] = sum(i in Instr) investment[i,p];` *// "investment" is a decision variable*
  is compiled into
  ```
  wealth[<p,ScenTree[p,scen]>] =
                  sum(i in Instr) investment[i,<p,ScenTree[p,scen]>];
  ```

- ```
  wealth[p+1] = sum (i in Instr)
          investment[i,p]*(1+ret[i,p]^market);
  ```
  is compiled into
  ```
    wealth[<p+1,ScenTree[p+1,scen]>] = sum (i in Instr)
        investment[i,<p,ScenTree[p,scen]>]*(1+ret[i,<p,ScenTree[p+1,scen]>]);
  ```

5.2 Compiling Chance Constraints

The chance constraints posted using a command of the form

```
 prob(<Constraint>) <ArithOp> <Expr>;
```

are compiled into a sum constraint of the form

```
 sum(scen in Scenarios)
    Probability[scen]*(<Constraint[scen]>) <ArithOp> <Expr>;
```

where `<Constraint[scen]>` is a compiled Stochastic OPL constraint in scenario `scen`, `Probability[scen]` the probability of scenario `scen` and `Scenarios` the set of all scenarios.

As an example of chance constraint compilation consider the following inventory constraints:

```
forall (p in Periods) Stock[p]+Order[p]-Demand[p] = Stock[p+1];
forall (p in Periods) Stock[p+1] >= 0;
```

These are inventory balance equations and non-negative stock constraints, respectively. In the case of stochastic demand, it is generally very expensive to follow a policy which guarantees no backlogging, i.e., meeting all customers' demand. Instead, generally, a target service level is introduced by the management and the complete demand satisfaction policy is relaxed. Hence, the inventory problem now becomes:

```
forall (p in Periods) Stock[p]+Order[p]-Demand[p] = Stock[p+1];
forall (p in Periods) prob(Stock[p+1] >= 0) >= ServLev;
```

The inventory balance equation, which is a hard constraint, can be compiled into its certainty equivalent form as explained in Sec.5.1. The compilation of chance constraints are done in a similar fashion, with the only exception of the introduction of weights, which are actually the probabilities of relevant scenarios. The service-level expression is compiled into the following OPL constraint:
```

```
forall (p in Periods)
  sum (scen in 1..nbNodes[p+1])
  (Stock[<p+1,ScenTree[p+1,scen]>] >= 0)*Probability[<p+1,scen>] >= ServLev;
```

Note that the bracketing of the inequality reifies the constraint so that it takes the value 1 if satisfied and 0 otherwise.

5.3 Compiling Objective Functions

The most common objective function type, the expected value function, is incorporated into Stochastic OPL with the reserved word `expected` and can be compiled into standard OPL the same way as `prob`:

```
maximize expected(<Expr>)
```

compiles into

```
maximize
   sum(scen in Scenarios)
      (<Expr[scen]>)*Probability[scen]
```

Another important objective function type, Markowitz's mean/variance model, `mv(<Expr>,`λ`)` or $E\{Q\} + \lambda E\{K\}$, which takes into account the tradeoff between the expected value of an objective function and its variability, can easily be expressed in the certainty equivalent form by employing the absolute deviation function of Sec.4.3:

```
minimize
  expected(Q) + λ*expected(K)
subject to {
   K - Q + expected(Q) ≥ 0;
   K + Q - expected(Q) ≥ 0;
   ... };
```

Other objective functions, namely,

```
minimize spread(profit)
maximize downside(profit)
minimize upside(cost)
```

which are supported by Stochastic OPL are compiled in a similar fashion.

This is by no means an exhaustive list but it gives an indication of the variety of optimization goals and the versatility of the Stochastic OPL system. New optimization goals can be incorporated into the system as needed.

6 Value of information and stochastic solutions

We can also easily provide the user with information about how much value is obtained if we were to know the value of stochastic variables. For example, in some situations it can be possible to wait for stochastic variables to realize their values. Alternatively, we can show how expensive it is to fix to a solution now that ignores future changes.

Consider a payoff table based on m possible decisions D_i for $i = 1, ..., m$ and n possible scenarios S_j for $j = 1, ..., n$. The payoff for decision D_i, if scenario S_j will occur, is a_{ij}. Suppose the probability that scenario S_j will occur is p_j. If the decision criterion is the expected payoff, then the best decision is the one that maximizes $\sum_{j=1}^{n} p_j a_{ij}$ and the solution is called "stochastic solution", SS.

There is a family of models, one for each scenario, and the weighted average of solutions for each scenario (solved by assuming that all data were already known) gives the expected "wait-and-see solution", WSS.

Consider the hypothetical situation that one knows ahead of time which scenario will occur. If such information is available then one may expect extra payoff with a non-negative value. The expected value of payoff can serve as an upper bound for the value of information, which is called the "expected value of perfect information" (EVPI). It is assumed that the probability that the perfect information will indicate that scenario S_j will occur is also p_j.

¿From the definition, the EVPI is the difference between the expected payoff calculated using the maximum a_{ij} for each scenario (WSS) and the expected payoff for the best decision (SS).

The expression for the EVPI is thus:

$$EVPI = WSS - SS = \sum_{j=1}^{n} p_j \max_{1 \le i \le m} \{a_{ij}\} - \max_{1 \le i \le m} \left\{ \sum_{j=1}^{n} p_j a_{ij} \right\}$$

This is therefore the most that should be spent in gathering information about the uncertain world.

For stochastic optimization problems, we may compute another statistics which quantify the importance of randomness. The "value of stochastic solution", VSS, measures the possible gain from solving the stochastic model that explicitly incorporates the distribution of random variables within the problem formulation.

Some models do not take into account the randomness of different uncertain parameters. They replace the uncertain parameters by their expected values and solve then the so called expected value problem. It means that only one scenario, namely the expected value scenario, is considered. In this case the solution to the expected value scenario will give an objective function value for the stochastic problem, which is called the "expected value solution", EVS.

The value of a stochastic solution (VSS) is then the difference between SS and EVS:

$$VSS = SS - EVS \geq 0.$$

This computes the benefit of knowing the distributions of the stochastic variables. It is a well-known fact in decision theory that the above relation is valid. This means that the objective function's expected value of the stochastic optimization problem will be better than the expected value of deterministic programming.

These statistics can easily be calculated using our framework. Such calculation requires the solution of n independent deterministic scenario problems to determine WSS and a single expected value problem to determine EVS.

7 Scenario reduction

Each scenario introduces new decision variables. For many practical problems, it is too expensive to compute all possible scenarios. How then can we replace a large, computationally intractable scenario tree with a small, tractable tree so that solving the problem over the small tree yields a solution not much different than the solution over the large tree?

We have implemented several techniques to reduce the number of scenarios. These scenario reduction algorithms determine a subset of scenarios and a redistribution of probabilities relative to the preserved scenarios. No requirements on the stochastic data process are imposed and therefore the concept is general. However, the reduction algorithms, depending on their sophistication, may require different types of data.

The simplest scenario reduction algorithm is to consider just a single scenario in which stochastic variables take their expected values. This is supported with the command:

```
scenario expected;
```

This is actually the aforementioned (see Sec.6) expected value problem. Among the other methods presented here, this is the most crude one.

The user may also be content to consider just a few of the most probable scenarios and ignore rare events. This method is referred as “mostlikely” in the rest of the paper. We support this with the command:

```
scenario top <Num>;
```

Another option is to use Monte Carlo sampling. The user can specify the number of scenarios to sample using a command of the form:

```
scenario sample <Num>;
```

The probability distributions of the stochastic variables is used to bias the construction of these scenarios.

We can also consider sampling methods which may converge faster than simple Monte Carlo sampling. For example, we implemented one of the best sampling methods from experimental design, and one of the best scenario reduction methods from operations research. Latin hypercube sampling (LHS) [17], ensures that a range of values for a variable are sampled. Suppose we want the sample size to be n. We divide the unit interval into n intervals, and sample a value for each stochastic variable whose cumulative probability occurs in each of these interval. We then construct n sample scenarios from these values, enforcing the condition that the samples use each value for each stochastic variable exactly once. More precisely, let $f_i(a)$ be the cumulative probability that X_i takes the value a or less, $P_i(j)$ be the jth element of a random permutation P_i of the integers $\{0, \ldots, n-1\}$, and r be a random number uniformly drawn from $[0, 1]$. Then, the jth Latin hypercube sample value for the stochastic variable X_i is:

$$f_i^{-1}(\frac{P_i(j) + r}{n})$$

However, it should be noted that the sample size n does not guarantee to produce a sample of n scenarios, since a single scenario may be chosen more than once due to, for example, the discreteness of the data. The command for LHS is

```
scenario lhs <Num>;
```

where `<Num>` denotes the number of non-overlapping intervals used with LHS.

Finally, we implemented a scenario reduction method used in stochastic programming due to Dupacova, Growe-Kuska and Romisch [4]. Dupacova et al. assume that the original probability measure P is discrete with finitely many scenarios and this probability measure is approximated by a probability measure Q of a smaller number of scenarios. In this case, the upper bound for the distance between P and Q is a Kantorovich functional. Then the upper bound represents the optimal value of a Monge-Kantorovich mass transportation problem. The Monge-Kantorovich mass transportation problem dates back to work by Monge in 1781 on how to optimally move material from one place to another, knowing only its initial and final spatial distributions, the cost being a prescribed function of the distance travelled by molecules of material [22], [23]. Dupacova et al. show that the Kantorovich functional of the original probability distribution P and the optimal reduced measure Q based on a given subset of scenarios of P as well as the optimal weights of Q can be computed without solving the Monge-Kantorovich problem. Their backward reduction algorithm for determining a subset of scenarios is given below.

Let n_T denote the number of stages of the optimization problem and n_s the number of scenarios. A scenario $\omega^{(i)}$, $i \in \{1, ..., n_s\}$, is defined as a sequence of nodes of the tree

$$\omega^{(i)} = (\eta_0^{(i)}, \eta_1^{(i)}, ..., \eta_{n_T}^{(i)}), \qquad i = 1, ..., n_s$$

For each node belong to scenario j on stage s, a vector $\mathbf{p}_s^{(j)} \in \mathbb{R}^{n_s^p}$ of parameters is given. The probability to get from $\eta_j^{(i)}$ to $\eta_{j+1}^{(i)}$ is denoted by $\pi_{j,j+1}^{(i)}$. Thus the probability for the whole scenario $\omega^{(i)}$ is given by

$$\pi^{(i)} = \prod_{j=0}^{n_T-1} \pi_{j,j+1}^{(i)}$$

The distance between two scenarios $\omega^{(i)}$ and $\omega^{(j)}$ is defined as

$$d(\omega^{(i)}, \omega^{(j)}) = \sqrt{\sum_{s=0}^{n_T} (\mathbf{p}_s^{(i)} - \mathbf{p}_s^{(j)})^2}$$

according to the Euclidean norm in the space of the parameter vectors.

The below scenario deletion procedure is applied iteratively, deleting one scenario in each iteration, until a given number of scenarios remains.

S1. Determine the scenario to be deleted: Remove scenario $\omega^{(s^*)}$, $s^* \in \{1, ..., n_s\}$ satisfying

$$\pi^{(s^*)} \min_{\bar{s} \neq s^*} d(\omega^{(\bar{s})}, \omega^{(s^*)}) = \min_{m,n \in \{1,...,n_s\}} \{\pi^{(m)} \min_{n \neq m} d(\omega^{(n)}, \omega^{(m)})\}$$

Hence, not only the distances, but also the probabilities of the scenarios are considered.

S2. Change the number of scenarios: $n_s := n_s - 1$.

S3. Change the probability of the scenario $\omega^{(\bar{s})}$, that is the nearest to $\omega^{(s^*)}$:

$$\pi^{(\bar{s})} := \pi^{(\bar{s})} + \pi^{(s^*)}$$

$$\pi^{(s^*)} := 0$$

All node probabilities are adjusted, as the sum of the probabilities of possible realizations at each node equals 1.

S4. If $n_s > N$ then go to step 1, where N is the desired number of scenarios remain.

This algorithm is incorporated into Stochastic OPL and can be called by

```
scenario DGR <Num>;
```

Dupacova et al. report power production planning problems on which this method offers 90% accuracy sampling 50% of the scenarios and 50% accuracy sampling just 2% of the scenarios.

8 Some examples

To illustrate the potential of this framework for decision making under uncertainty, we now describe a wide range of problems that we have modelled. In each problem, we illustrate the effectiveness of different scenario reduction techniques.

8.1 Portfolio Diversification

This portfolio diversification problem of [2] can be modelled as a stochastic COP. Suppose we have \$$P$ to invest in any of I investments and we wish to exceed a wealth of \$$G$ after t investment periods. To calculate the utility, we suppose that exceeding \$$G$ is equivalent to an income of q% of the excess while not meeting the goal is equivalent to borrowing at a cost r% of the amount short. This defines a concave utility function for $r > q$. The uncertainty in this problem is the rate of return, which is a random variable, on each investment in each period. The objective is to determine the optimal investment strategy, which maximizes the investor's expected utility. A mathematical formulation of the problem, and corresponding stochastic and certainty-equivalent OPL representations are given in the appendix.

The test problem has 8 stages, in which the number of states are [5,4,4,3,3,2,2,2], and 5760 scenarios. The CP model has 27 decision variables and 18 constraints for one scenario, and 33,438 decision variables and 22,292 constraints for 5,760 scenarios. To compare the effectiveness of different scenario reduction algorithms, we adopt a two step procedure. In the first step, the scenario reduced problem is solved and the first period's decision is observed. We then solve the full-size (non scenario reduced) problem to optimality with this first decision fixed. The difference between the objective values of these two solutions is normalized by the range [optimal solution, observed worst solution] to give a normalized error for committing to the scenario reduced first decision. In Fig. 2, we see that Dupacova et al's algorithm is very effective, that Latin hypercube sampling is a small distance behind, and both are far ahead of the most likely scenario method (which requires approximately half the scenarios before the first decision is made correctly).

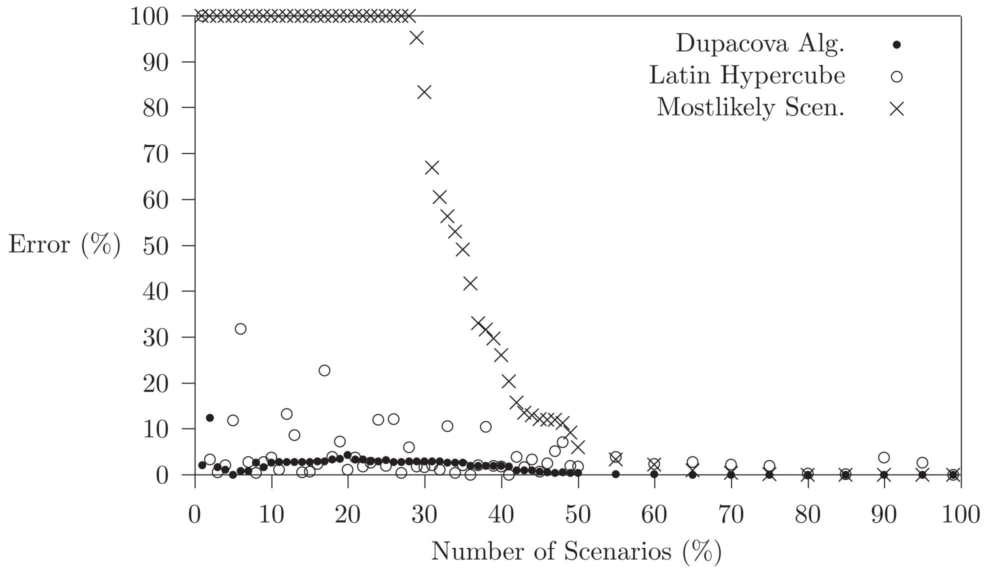

Figure 2: Portfolio Diversification – I

A smaller version of the above problem is designed with 4 stages, [5,4,4,3] states in each stage and hence 240 scenarios in total. The CP model has 15 decision variables and 10 constraints for one scenario, and 1,038 decision variables and 692 constraints for 240 scenarios. Fig.3 shows that Dupacova et al. algorithm requires approximately one third the scenarios before the first decision is made correctly. It is interesting to see that the general performance of scenario reduction algorithms has deteriorated in the 4-stage case. This is mainly due to the fact that longer the planning horizon, the better the chance of recovery from an early mistake.

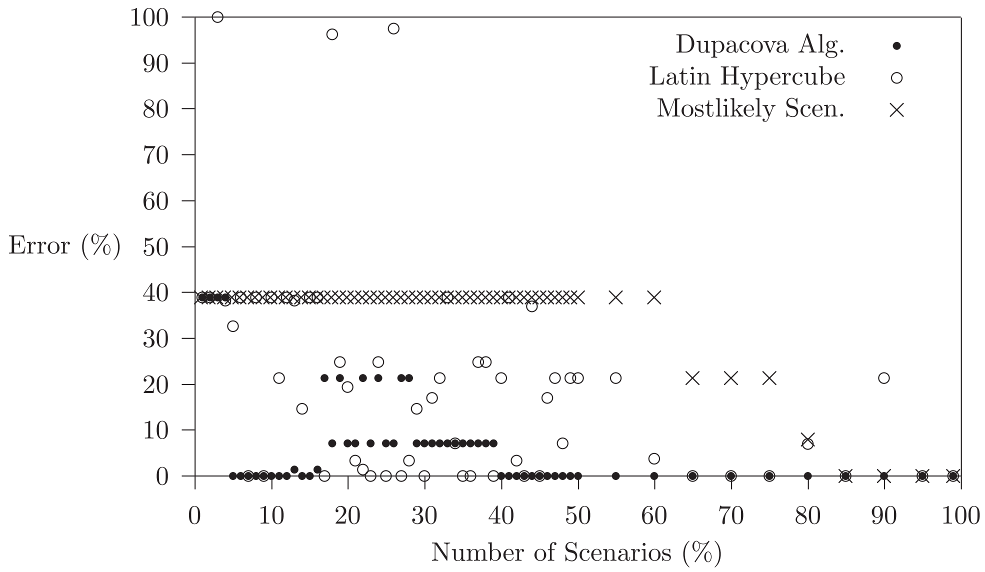

Figure 3: Portfolio Diversification – II

8.2 Agricultural Planning

Farmers must deal with uncertainty since weather and many other factors affect crop yields. In this example (also taken from [2]), we must decide on how many acres of his fields to devote to various crops before the planting season. A certain amount of each crop is required for cattle feed, which can be purchased from a wholesaler if not raised on the farm. Any crop in excess of cattle feed can be sold up to the EU quota; any amount in excess of this quota will be sold at a low price. Crop yields are uncertain, depending upon weather conditions during the growing season. This test problem (Agricultural Planning – I) has 4 stages and 10,000 scenarios. The CP model has 55 decision variables and 30 constraints for one scenario, and 163,324 decision variables and 116,661 constraints for 10,000 scenarios. In Fig. 4, we again see that Dupacova et al's algorithm and Latin hypercube sampling are very effective, and both are far ahead of the most likely scenario method. Fig.5 shows the results for a smaller instance (Agricultural Planning – II), with 1 stage and 10 scenarios only. The CP model has 16 decision variables and 9 constraints for one scenario, and 124 decision variables and 81 constraints for 10 scenarios. The adverse effect of the shorter planning horizon on the effectiveness of scenario reduction techniques is also observed in this case.

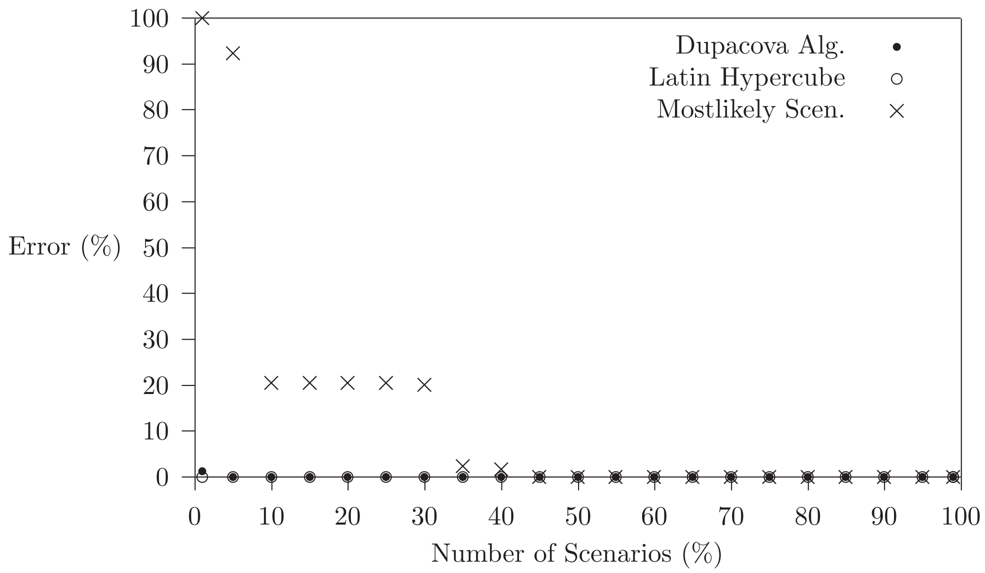

Figure 4: Agricultural Planning – I

8.3 Production/Inventory Management

Uncertainty plays a major role in production and inventory management. In this simplified production/inventory planning example, there is a single product, a single stocking point, production capacity constraints and stochastic demand. The objective is to find the minimum expected cost policy. The cost components taken into account are holding costs, backlogging costs, fixed replenishment (or setup) costs and unit production costs. The optimal policy gives the timing of the replenishments as well as the order-up-to-levels. Hence, the exact order quantity can be known only after the realization of the demand, using the scenario dependent order-up-to-level decisions. This test problem has 5 stages

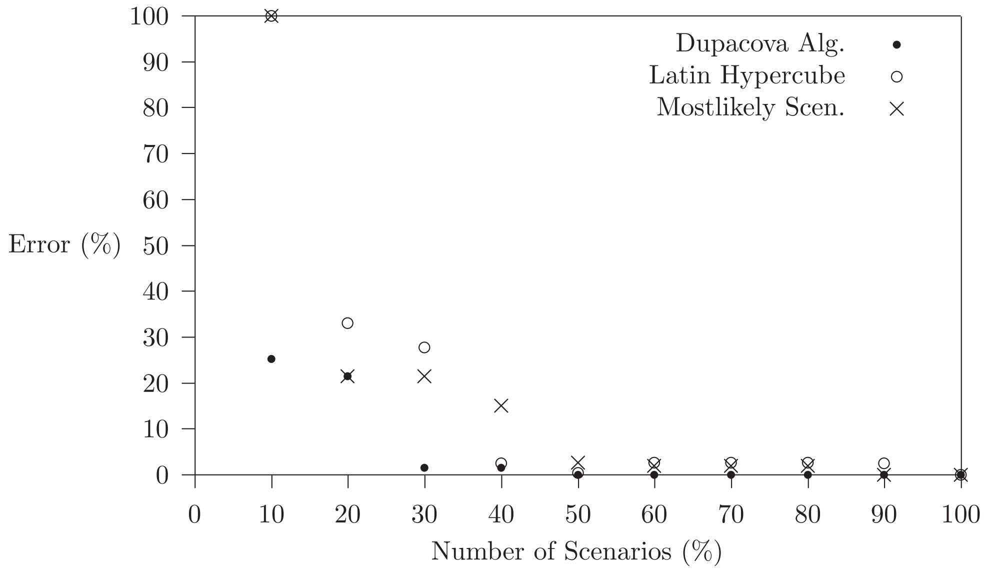

Figure 5: Agricultural Planning – II

(4 states in each) and 1,024 scenarios. The CP model has 26 decision variables and 21 constraints for one scenario, and 4,775 decision variables and 3,411 constraints for 1,024 scenarios. In Fig. 6, we again see that Dupacova et al's algorithm and Latin hypercube sampling are very effective, but both are now only a small distance ahead of the most likely scenario method.

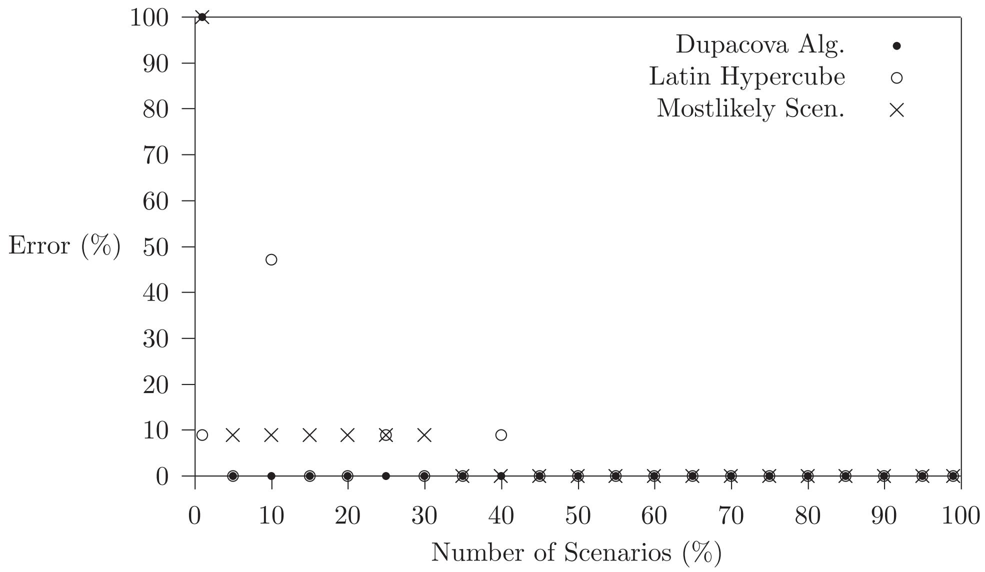

Figure 6: Production/Inventory Management (Full Cost)

The study of stochastic production/inventory problems is often divided into two broad groups: the full cost model and the partial cost model with a service level constraint. An example of the full cost model is given above. The service level approach introduces a service level constraint in place of the shortage cost, where the service level refers to the availability of stock in an expected sense.

A certainty equivalent MIP formulation of this problem, under non-stationary stochastic continuous demand assumption, is given by [26]. The same problem, but under discrete demand assumption, is tackled here using the Stochastic CP framework and the results for the scenario reduction algorithms are given in Fig. 7.

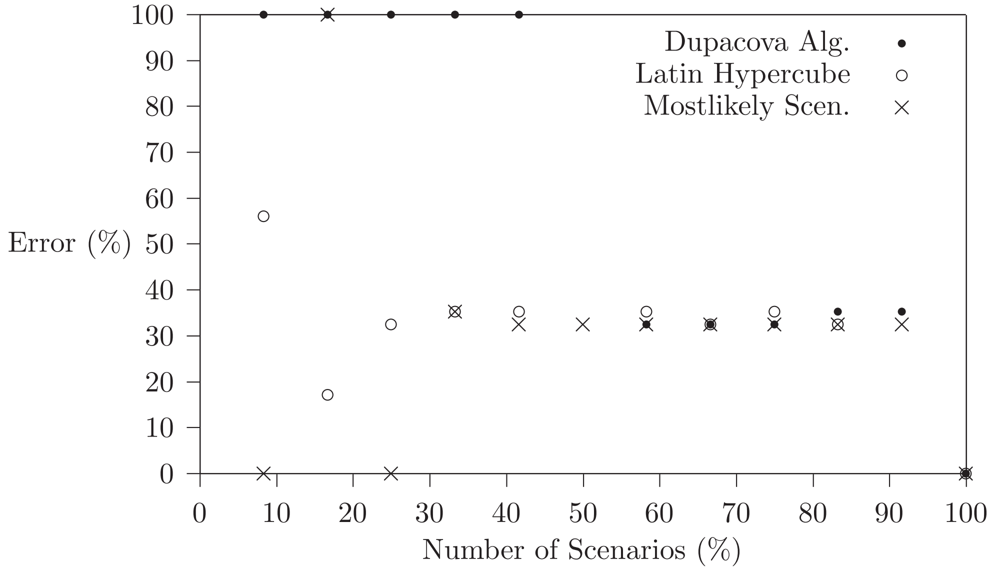

Figure 7: Production/Inventory Management (Service Level)

The performances of scenario reduction algorithms exhibit a completely different pattern in the service-level version of the production/inventory example. In contrast to our previous observations, now the "mostlikely" method outperforms other two methods. In only 1 case out of a total of 11 cases the "mostlikely" method is surpassed by LHS. It should also be noted that Dupacova et al. method gives an infeasible solution in one case and LHS in two cases, whereas the "mostlikely" method consistently produces feasible solutions. An explanation for this outcome lies in the very nature of the problem under consideration and the probability redistribution mechanism of the scenario reduction algorithms. In Dupacova et al., in each iteration two closest scenarios are chosen and reduced to one. The deleted scenario's probability is added to the preserved one's probability. Likewise, in LHS, probabilities are redistributed in accordance with the outcome of simulation experiments. These two methods modify the existing probability structure substantially, whereas the "mostlikely" method chooses the most probable scenarios and then only normalises their probabilities, which is less of a radical change compared to other two methods. Therefore, the better performance of the "mostlikely" method hinges on these aspects of scenario reduction algorithms in conjunction with those of chance-constrained problems where probabilities are not only a factor that affects the expected value of the objective function but feasibility itself.

9 Robust solutions

Inspired by robust optimization methods in operations research [13], we can also find robust solutions to stochastic constraint programs. That is, solutions in which similar decisions are made in different scenarios. It will often be impossible or undesirable for all decision variables to be robust. We therefore identify those decision variables whose values we wish to be identical across scenarios using commands of the form:

```
robust <Var>;
```

For example, in production/inventory problem of Sec.8.3 the decision variables "order-up-to-levels" and "replenishment periods" can be declared as robust variables. The values of these two sets of decision variables are then fixed at the beginning of the planning horizon giving a static policy. A robust solution dampens the nervousness of the solution, an area of very active research in production/inventory management. As the expected cost of the robust solution is always higher, the tradeoff between nervousness and cost may have to be taken into account.

According to [18], the optimal solution of the program will be robust with respect to optimality if it remains close to optimal for any realization of the scenario $\omega \in \Omega$. It is then termed "solution robust". The objective function can be written in the form, $\min \sigma(x, y_{\omega\in\Omega})$ where x denotes the deterministic decision variables, $y_{\omega\in\Omega}$ is a set of control variables for each scenario.

There is not a unique form of the above function. As discussed in Sec.4.3, one typical form can be the expected value criterion, $\sigma(.) = \sum_{\omega\in\Omega} p_\omega Q_\omega$, in which the objective function of a model becomes a random variable taking the value Q_ω with probability p_ω. Another common form is the worst-case criterion, $\sigma(.) = \max_{\omega\in\Omega} Q_\omega$.

Mulvey, Vanderbei and Zenios point out that the expected value and the worst-case functions are special cases in robust optimization, and the tradeoff between mean value and its variability is a novelty of the robust optimization formulation. However, as discussed in Sec.4.3, Markowitz's mean/variance model provides such a framework.

To demonstrate the concept and the use of robustness in stochastic constraint programming, we consider a production/inventory planning problem with demand data provided in Table 2, a production capacity of 40 units/period, and stationary costs: production/purchasing costs \$2/unit, fixed ordering (or setup) costs \$50/replenishment, inventory holding costs \$1/unit/period, backlogging costs \$5/unit/period.

	D_1	D_2	D_3	D_4	$\Pr\{D_1\}$	$\Pr\{D_2\}$	$\Pr\{D_3\}$	$\Pr\{D_4\}$
Period 1	8	10	12	14	0.20	0.20	0.40	0.20
Period 2	15	18	21	24	0.30	0.40	0.20	0.10
Period 3	15	20	23	26	0.10	0.20	0.60	0.10
Period 4	10	15	20	22	0.50	0.30	0.10	0.10
Period 5	12	18	20	24	0.30	0.50	0.10	0.10

Table 2: Demand data

Stochastic-CP model, with `minimize expected(cost)`, gives the following solution: In Period 1, inventory is raised to 33; no replenishment is planned for Period 2; in Period 3, a replenishment is planned with a scenario dependent order-up-to-level varying between 35 to 47; in Period 4, only in one scenario out of 64 there is a replenishment; in the final period, there are replenishments with various order-up-to-levels in 36 scenarios out of 256. The expected total cost of following the optimal policy is \$351.61.

To have a static policy, order-up-to-levels and replenishment decisions are declared as

```
robust int+ Order_up_to[Period] in 0..maxint;
```

and

```
robust int Replenish[Period] in 0..1;
```

which give a less nervous policy. Now the best replenishment policy is to have replenishment every period, Periods 1-5, with an order-up-to-level [14,21,23,20,18] respectively. The expected total cost of this scheme is \$439.70.

The solution robust policy, assuming a tradeoff constant $\lambda = 1.0$, is determined using the Stochastic OPL objective function `minimize mv(cost,λ)`. The optimal replenishment plan is now: in Period 1, the order-up-to-level is 35; no replenishment is planned for Period 2; in Period 3, depending on demand, order-up-to-level takes a value between 37 and 51; no replenishment in Period 4; and finally, in Period 5, there are replenishments in 16 scenarios. The expected total cost of following the solution robust policy is \$353.06.

In Fig.8, we see that as one would expect, an increase in λ, which actually points to a decrease in the objective value uncertainty, causes an increase in the expected total production and inventory costs.

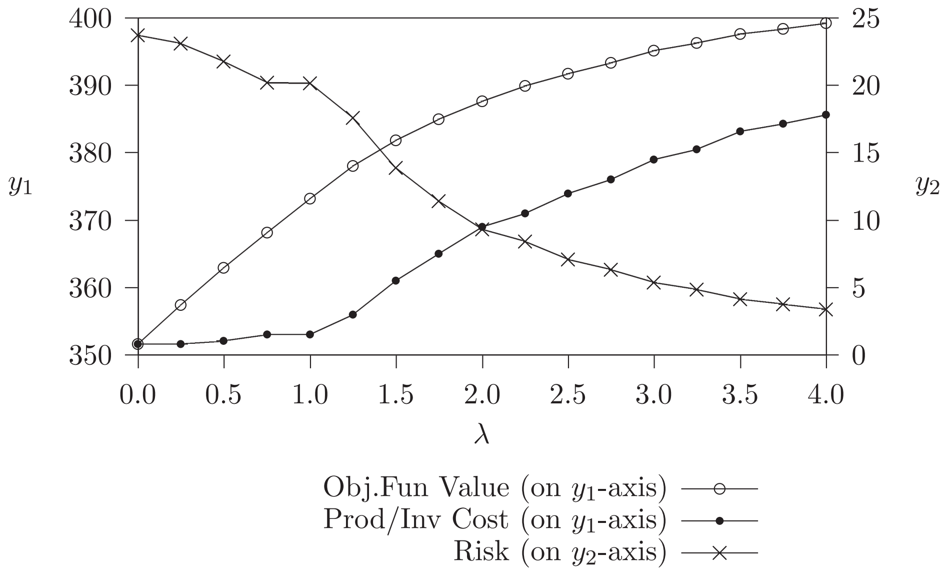

Figure 8: λ vs. Risk, Prod/Inv Cost, Obj.Func Value

10 Related work in decision making under uncertainty

Stochastic constraint programs are closely related to Markov decision problems (MDPs). An MDP model consists of a set of states, a set of actions, a state transition function which gives the probability of moving between two states as a result of a given action, and a reward function. A solution to an MDP is a policy, which specifies the best action to take in each possible state. MDPs have been very influential in AI of late for dealing with situations involving reasoning under uncertainty [21]. Stochastic constraint programs can model problems which lack the Markov property that the next state and reward depend only on the previous state and action taken. To represent a stochastic constraint program in which the current decision depends on all earlier decisions would require an MDP with an exponential number of states. Stochastic constraint optimization can also be used to model more complex reward functions than the (discounted) sum of individual rewards. Another significant difference is that stochastic constraint programs by using a scenario-based interpretation can immediately call upon complex and powerful constraint propagation techniques.

Stochastic constraint programs are also closely related to influence diagrams. Influence diagrams are Bayesian networks in which the chance nodes are augmented with decision and utility nodes [19]. The usual aim is to maximize the sum of the expected utilities. Chance nodes in an influence diagram correspond to stochastic variables in a stochastic constraint program, whilst decision nodes correspond to decision variables. The utility nodes correspond to the cost function in a stochastic constraint optimization problem. However, reasoning about stochastic constraint programs is likely to be easier than about influence diagrams. First, the probabilistic aspect of a stochastic constraint program is simple and decomposable as there are only unary marginal probabilities. Second, the dependencies between decision variables and stochastic variables are represented by declarative constraints. We can therefore borrow from traditional constraint satisfaction and optimization powerful algorithmic techniques like branch and bound, constraint propagation and nogood recording. As a result, if a problem can be modelled within the more restricted format of a stochastic constraint program, we hope to be able to reason about it more efficiently.

11 Related work in constraints

Stochastic constraint programming was inspired by both stochastic integer programming and stochastic satisfiability [15]. It is designed to take advantage of some of the best features of each framework. For example, we are able to write expressive models using non-linear and global constraints, and to exploit efficient constraint propagation algorithms. In operations research, scenarios are used in stochastic programming. Indeed, the scenario reduction techniques of Dupacova, Growe-Kuska and Romisch [4] implemented here are borrowed directly from stochastic programming.

Mixed constraint satisfaction [7] is closely related to one stage stochastic constraint satisfaction. In a mixed CSP, the decision variables are set after the stochastic variables are given random values. In addition, the random values are chosen uniformly. In the case of full observability, the aim is to find conditional values for the decision variables in a mixed CSP so that we satisfy all possible worlds. In the case of no observability, the aim is to find values for the decision variables in a mixed CSP so that we satisfy as many possible worlds. An earlier constraint satisfaction model for decision making under uncertainty [6] also included a probability distribution over the space of possible worlds.

Constraint satisfaction has been extended to include probabilistic preferences on the values assigned to variables [25]. Associated with the values for each variable is a probability distribution. A "best" solution to the constraint satisfaction problem is then found. This may be the maximum probability solution (which satisfies the constraints and is most probable), or the maximum expected overlap solution (which is most like the true solution). The latter can be viewed as the solution which has the maximum expected overlap with one generated at random using the probability distribution. The maximum expected overlap solution could be found by solving a suitable one stage stochastic constraint optimization problem.

Branching constraint satisfaction [8] models problems in which there is uncertainty in the number of variables. For example, we can model a nurse rostering problem by assigning shifts to nurses. Branching constraint satisfaction then allows us to deal with the uncertainty in which nurses are available for duty. We can represent such problems with a stochastic CSP with a stochastic 0/1 variable for each nurse representing their availability.

A number of extensions of the traditional constraint satisfaction problem model constraints that are uncertain, probabilistic or not necessarily satisfied (see, for instance, [5, 3, 28]). In partial constraint satisfaction we maximize the number of constraints satisfied [9]. As a second example, in probabilistic constraint satisfaction each constraint has a certain probability independent of all other probabilities of being part of the problem [5]. As a third example, both valued and semi-ring based constraint satisfaction [3] generalizes probabilistic constraint satisfaction as well as a number of other frameworks. In semi-ring based constraint satisfaction, a value is associated with each tuple in a constraint, whilst in valued constraint satisfaction, a value is associated with each constraint. As a fourth example, the certainty closure model [28] permits constraints to have parameters whose values are uncertain. This differs from stochastic constraint programming in three significant ways. First, stochastic variables come with probability distributions in our framework, whilst the uncertain parameters in the certainty closure model take any of their possible values. Second, we find a policy which may react differently according to the values taken by the stochastic variables, whilst the certainty closure model aims to find the decision space within which all possible solutions must be contained. Third, stochastic constraint programs can have multiple stages whilst certainty closure models are essentially one stage. Stochastic constraint programming can easily be combined with most of these techniques. For example, we can define stochastic partial constraint satisfaction in which we maximize the number of satisfied constraints, or stochastic probabilistic constraint satisfaction in which each constraint has an associated probability of being in the problem.

12 Conclusions

We have described stochastic constraint programming, an extension of constraint programming to deal with both decision variables (which we can set) and stochastic variables (which follow some probability distribution). This framework is designed to take advantage of the best features of traditional constraint satisfaction, stochastic integer programming, and stochastic satisfiability. It can be used to model a wide variety of decision problems involving uncertainty and probability. We have provided a semantics for stochastic constraint programs based on scenarios. We have shown how to compile stochastic constraint programs down into conventional (non-stochastic) constraint programs. We can therefore call upon the full power of existing constraint solvers without any modification. We have also described a number of techniques to reduce the number of scenarios, and to generate robust solutions.

We have implemented this framework for decision making under uncertainty in a language called stochastic OPL. This is an extension of the OPL constraint modelling language [10]. To illustrate the potential of this framework, we have modelled a wide range of problems in areas as diverse as finance, agriculture and production. There are many directions for future work. For example, we want to allow the user to define a limited set of scenarios that are representative of the whole. As a second example, we want to explore more sophisticated notions of solution robustness (e.g. limiting the range of values used by a decision variable).

Acknowledgements

S. Armagan Tarim is supported by Science Foundation Ireland under Grant No. 03/CE3/I405 as part of the Centre for Telecommunications Value-Chain-Driven Research (CTVR) and Grant No. 00/PI.1/C075, and by EPSRC under GR/R30792. Suresh Manandhar is supported by EPSRC under GR/R30792. We thank the other members of the 4C Lab and APES Research Group.

Appendix

In this appendix, we present a mathematical formulation of the portfolio diversification problem of Section 8.1. We also give corresponding stochastic and certainty-equivalent OPL representations.

Assume that, we have \$$W$ to invest, denoted by decision variables $inv[i]$, in any of $i \in I$ instruments (stocks, bonds, etc.) over a planning horizon of N periods,

$$\begin{aligned}
&wealth[1] = W \\
&wealth[p] = \sum_{i \in I} inv[i,p], \quad p \in \{1,...,N\}
\end{aligned}$$

We wish to exceed a wealth of \$$G$ at the end of the planning horizon. To calculate the utility, we suppose that exceeding \$$G$ is equivalent to an income of q% of the excess while not meeting the goal is equivalent to borrowing at a cost r% of the amount short. So, we can write,

$$\begin{aligned}
&\max E(q \times pos - r \times neg) \\
&pos - neg = wealth[N+1] - G \\
&pos, neg \in \mathbb{Z}^{0,+}.
\end{aligned}$$

The uncertainty in this problem is the rate of return, $return$, which is a random variable, on each investment in each period.

$$wealth[p+1] \leq \sum_{i \in I} inv[i,p](1 + return[i,p]), \quad p \in \{1,...,N\}$$

A stochastic CP model for the above formulation, written in stochastic OPL, is as follows:

```
stoch market[Period] = ...;
enum I ...;
int N = ...;
range Period [1..N];
range Period_M [1..N+1];
float return[I,Period]^market = ...;
var int+ inv[I,Period] in 0..200;
```

```
var int wealth[Period_M] in 100..200;
var int+ pos in 0..50;
var int+ neg in 0..50;

maximize expected(pos - 4*neg)
subject to {
wealth[1] = 100;
pos - neg = wealth[N+1] - 150;
forall (p in Period) wealth[p] = sum (i in I) inv[i,p];
forall (p in Period) wealth[p+1] <= sum (i in I) inv[i,p]*(1+return[i,p]);
};
```

A possible corresponding input data is

```
I = {stock,bond};
N = 3;
market =[<0.0(0.5),0.0(0.5)>,<0.0(0.5),0.0(0.5)>,<0.0(0.5),0.0(0.5)>];
return = [[<0.25,0.06>,<0.25,0.06>,<0.25,0.06>],[<0.14,0.12>,<0.14,0.12>,<0.14,0.12>]];
```

The stochastic OPL model can then be compiled down into the following certainty-equivalent OPL model:

```
enum I ...;
int N = ...;
range Period_Ext [1..N+1];
range Period [1..N];
range Period_M [1..N+1];
int nbStates[Period] = ...;
int nbNodes[Period_Ext] = ...;
int+ ScenTree[Period_Ext,1..nbNodes[N+1]] = ...;
struct TreeType {Period_Ext stage; int state; };
{TreeType} States ={<stage,state>|stage in Period & state in 1..nbStates[stage]};
{TreeType} Nodes ={<stage,state>|stage in Period & state in 1..nbNodes[stage]};
{TreeType} Nodes_Ext ={<stage,state>|stage in Period_Ext & state in 1..nbNodes[stage]};
{TreeType} Nodes_M ={<stage,state>|stage in Period_M & state in 1..nbNodes[stage]};
float Probability[Nodes_Ext] = ...;
float market[States] = ...;
float return[I,States] = ... ;
var int+ inv[I,Nodes] in 0..maxint;
var int wealth[Nodes_M] in 0..maxint;
var int+ pos[1..nbNodes[N+1]] in 0..maxint;
var int+ neg[1..nbNodes[N+1]] in 0..maxint;

maximize sum(scen in 1..nbNodes[N+1]) (pos[scen]-4*neg[scen])*Probability[<N+1,scen>]
```

```
subject to {
forall(scen in 1..nbNodes[N+1]) {
wealth[<1,ScenTree[1,scen]>] = 100;
pos[scen]-neg[scen] = wealth[<N+1,ScenTree[N+1,scen]>]-150;
forall(p in Period) wealth[<p,ScenTree[p,scen]>] =
    sum(i in I) inv[i,<p,ScenTree[p,scen]>];
forall(p in Period) wealth[<p+1,ScenTree[p+1,scen]>] <=
    sum(i in I) inv[i,<p,ScenTree[p,scen]>]*(1+return[i,<p,ScenTree[p+1,scen]>]);
};
};
```

The corresponding problem data is also compiled into

```
I={stock,bond};
N=3;
nbStates = [2, 4, 8];
nbNodes = [1, 2, 4, 8];
market = [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ];
return = [
[ 0.25 0.06 0.25 0.06 0.25 0.06 0.25 0.06 0.25 0.06 0.25 0.06 0.25 0.06 ]
[ 0.14 0.12 0.14 0.12 0.14 0.12 0.14 0.12 0.14 0.12 0.14 0.12 0.14 0.12 ]];
Probability = [1.0 0.5 0.5 0.25 0.25 0.25 0.25 0.125 0.125 0.125
0.125 0.125 0.125 0.125 0.125 ];
ScenTree = [
[ 1 1 1 1 1 1 1 1 ],
[ 1 1 1 1 2 2 2 2 ],
[ 1 1 2 2 3 3 4 4 ],
[ 1 2 3 4 5 6 7 8 ]];
```

This problem is solved to optimality and the obtained investment plan is depicted in Fig.9. S and B denotes the amounts that should be invested in stocks and bonds, respectively.

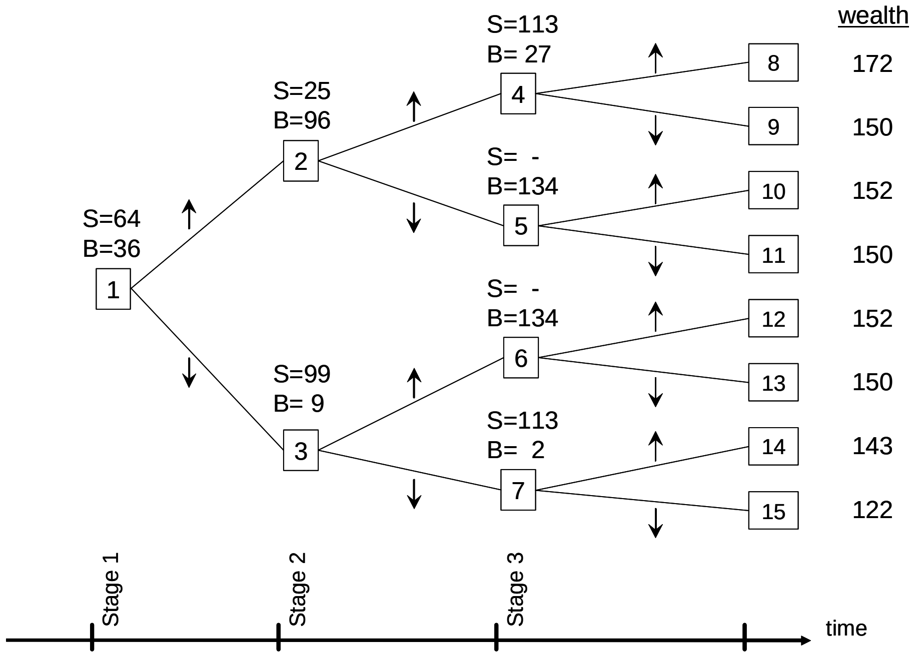

Figure 9: The optimal investment plan

References

[1] M. Allais. (1953). Le comportement de l'homme rationel devant le risque: critique des axioms et postulates de l'cole americaine. *Econometrica*, 21:503–546.

[2] J. R. Birge and F. Louveaux. (1997). *Introduction to Stochastic Programming.* Springer-Verlag, New York.

[3] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie. (1996). Semi-ring based CSPs and valued CSPs: Basic properties and comparison. In M. Jample, E. Freuder, and M. Maher, editors, *Over-Constrained Systems*, pages 111–150. Springer-Verlag. LNCS 1106.

[4] J. Dupacova, N. Growe-Kuska, and W. Romisch. (2003). Scenario reduction in stochastic programming: an approach using probability metrics. *Mathematical Programming*, Series A 95:493–511.

[5] H. Fargier and J. Lang. (1993). Uncertainty in constraint satisfaction problems: a probabilistic approach. In *Proceedings of ECSQARU*. Springer-Verlag. LNCS 747.

[6] H. Fargier, J. Lang, R. Martin-Clouaire, and T. Schiex. (1995). A constraint satisfaction framework for decision under uncertainty. In *Proceedings of the 11th International Conference on Uncertainty in Artificial Intelligence.*

[7] H. Fargier, J. Lang, and T. Schiex. (1996). Mixed constraint satisfaction: a framework for decision problems under in complete information. In *Proceedings of the 13th National Conference on Artificial Intelligence.* American Association for Artificial Intelligence.

[8] D. Fowler and K. Brown. (2000) Branching constraint satisfaction problems for solutions robust under likely changes. In *Proceedings of 6th International Conference on Principles and Practices of Constraint Programming.* Springer-Verlag.

[9] E. Freuder and R. Wallace. (1992). Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70.

[10] P. Van Hentenryck, L. Michel, L. Perron, and J-C. Regin. (1999). Constraint programming in OPL. In G. Nadathur, editor, *Principles and Practice of Declarative Programming*, LNCS 1702, pages 97–116. Springer-Verlag.

[11] H. Konno and H. Yamazaki. (1991). Mean-absolute deviation portfolio optimization model for its applications to Tokyo stock market. *Management Science*, 37:519–531.

[12] H. Konno. (1990). Piecewise linear risk functions and portfolio optimization. *Journal of the Operations Research Society of Japan*, 33:139–156.

[13] P. Kouvelis and G. Yu. (1996) *Robust Discrete Optimization and Its Applications.* Nonconvex optimization and its applications: volume 14. Kluwer.

[14] M. L. Littman, J. Goldsmith, and M. Mundhenk. (1998). The computational complexity of probabilistic plan existence and evaluation. *Journal of Artificial Intelligence Research*, 9:1–36.

[15] M. L. Littman, S. M. Majercik, and T. Pitassi. (2001). Stochastic boolean satisfiability. *Journal of Automated Reasoning.* 27:251–296.

[16] H. M. Markowitz. (1952). Portfolio selection. *Journal of Finance*, 7:77–91.

[17] M. D. McKay, R. J. Beckman, and W. J. Conover. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245.

[18] J. M. Mulvey, R. J. Vanderbei, and S. A. Zenios. (1995). Robust optimization of large-scale systems. *Operations Research*, 43:264–281.

[19] R. M. Oliver and J. Q. Smith. (1990). *Influence Diagrams, Belief Nets and Decision Analysis.* John Wiley and Sons.

[20] L. Proll and B. M. Smith. (1998). Integer Linear Programming and Constraint Programming Approaches to a Template Design Problem. *INFORMS Journal of Computing*, 10:265–275.

[21] M. L. Puterman. (1994). *Markov decision processes: discrete stochastic dynamic programming.* John Wiley and Sons.

[22] S. Rachev and L. Ruschendorf. (1998). *Mass Transportation Problems, Vol.I: Theory.* Springer.

[23] S. Rachev and L. Ruschendorf. (1998). *Mass Transportation Problems, Vol.II: Applications.* Springer.

[24] A. Ruszczynski and A. Shapiro. (2003). *Stochastic Programming, Handbooks in OR/MS, Vol:10.* Elsevier Science B.V.

[25] N. Shazeer, M. Littman, and G. Keim. (1999). Constraint satisfaction with probabilistic preferences on variable values. In *Proceedings of the 16th National Conference on Artificial Intelligence*. American Association for Artificial Intelligence.

[26] S. A. Tarim and B. G. Kingsman. (2004). The stochastic dynamic production/inventory lot–sizing problem with service–level constraints. *International Journal of Production Economics*, 88:105–119.

[27] T. Walsh. (2002). Stochastic Constraint Programming. in *Proceedings of ECAI-2002*.

[28] N. Yorke-Smith and C. Gervet. (2003). Certainty closure: A framework for reliable constraint reasoning with uncertainty. In F. Rossi, editor, *Proceedings of 9th International Conference on Principles and Practice of Constraint Programming (CP2003)*. Springer, Berlin.